RL-TR-97-110
Interim Report
October 1997

A SURVEY OF FAULT SIMULATION, FAULT GRADING AND TEST PATTERN GENERATION TECHNIQUES WITH EMPHASIS ON THE FEASIBILITY OF VHDL BASED FAULT SIMULATION

University of Virginia

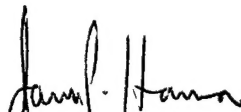Barry W. Johnson, D. Todd Smith, and Todd A. DeLong

19980223 111

DTIC QUALITY INSPECTED 3

**Rome Laboratory
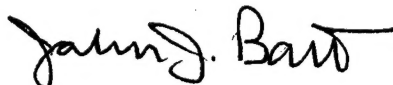Air Force Materiel Command
Rome, New York**

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-110 has been reviewed and is approved for publication.

APPROVED: *James P. Hanna*

JAMES P. HANNA
Project Engineer

FOR THE DIRECTOR: *John J. Bart*

JOHN J. BART, Chief Scientist
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/ERDD. 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED	
	October 1997	Interim	Sep 95 - Apr 97

4. TITLE AND SUBTITLE

A SURVEY OF FAULT SIMULATION, FAULT GRADING AND TEST PATTERN GENERATION TECHNIQUES WITH EMPHASIS ON THE FEASIBILITY OF VHDL BASED FAULT SIMULATION

6. AUTHOR(S)

B. W. Johnson, D.T. Smith, and T.A. DeLong

5. FUNDING NUMBERS

C   - F30602-95-C-0220
PE - 62702F
PR - 2338
TA - 01
WU - 9A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Virginia
Thornton Hall
Charlottesville VA 22903-2442

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory/ERDD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-110

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: James P. Hanna/ERDD/(315) 330-3473

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The primary purpose of this report is to determine the state-of-the-art for fault simulators which are used to estimate the test coverage for the DUT. It is envisioned that the state-of-the-art survey will be used to assist in defining the fault simulation techniques which are applicable to VHDL models. The goal is to fully understand the current fault simulation state-of-the-art so that existing techniques can be used to assist in the design of a VHDL-based fault simulation tool. One attribute which defines a VHDL-based simulator is that a VHDL compliant simulator is used to simulate the faulty device. Hierarchical serial fault simulation and hierarchical concurrent fault simulation are two techniques which can be used to develop a VHDL-based fault simulator. The state-of-the-art for fault grading techniques along with an overview of TPG methods is also provided in this report. While fault simulation is the main focus of this report, fault grading and TPG are included to completely describe the test generation, fault simulation, and fault grading process. It is important to realize that fault simulation is a means to assist TPG and estimate fault coverage via fault grading. The desired goal for a tool set is to contain a fault simulation technique which seamlessly augments the TPG process and performs fault grading in an efficient fashion.

14. SUBJECT TERMS

VHDL, Fault Simulation, WAVES, Test, Digital electronics, ATPG

15. NUMBER OF PAGES

134

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

1. Executive Summary

The design of high quality systems with a short time to market is becoming more prevalent. One part of the overall system quality assurance process is the testing of the manufactured system for design defects. A test development strategy to reveal faulty systems at the time of manufacture is required to assist the designer in demonstrating overall system quality. The test development process for digital systems/components is well established. The digital test development process can be subdivided into three stages: (1) Test Pattern Generation (TPG), (2) fault simulation, and (3) fault grading. The TPG process develops a set of test patterns which are applied to the Device Under Test (DUT) at the time of manufacture. Fault simulation is used to simulate faulty DUTs to determine if the test pattern set detects the faulty DUT. Conversely, fault grading takes the fault detection data provided by fault simulation to estimate the conditional probability of detecting a faulty device given that the device is faulty.

The primary purpose of this report is to determine the state-of-the-art for fault simulators which are used to estimate the test coverage for the DUT. It is envisioned that the state-of-the-art survey will be used to assist in defining the fault simulation techniques which are applicable to VHDL models. The goal is to fully understand the current fault simulation state-of-the-art so that existing techniques can be used to assist in the design of a VHDL-based fault simulation tool. One attribute which defines a VHDL-based fault simulator is that a VHDL compliant simulator is used to simulate the faulty device. Hierarchical serial fault simulation and hierarchical concurrent fault simulation are two techniques which can be used to develop a VHDL-based fault simulator.

The state-of-the-art for fault grading techniques along with an overview of TPG methods is also provided in this report. While fault simulation is the main focus of this report, fault grading and TPG are included to completely describe the test generation, fault simulation, and fault grading process. It is important to realize that fault simulation is a means to assist TPG and estimate fault coverage via fault grading. The desired goal for a tool set is to contain a fault simulation technique which seamlessly augments the TPG process and performs fault grading in an efficient fashion.

This report is organized into ten major sections. Following the Section 2 introduction, Section 3 provides background concerning fault simulation concepts. A high-level summary of the techniques covered in this report is presented in Section 4. Section 5 provides an overview of uni-processor based fault simulation techniques. Parallel processor fault simulation techniques are described in Section 6. A review of the use of hardware accelerators to achieve fast fault simulation is presented in Section 7. An overview of existing fault grading techniques is included as Section 8. Likewise, Section 9 provides a brief overview of test pattern generation techniques. An analysis of the applicability of the presented fault simulation methods for use in VHDL based fault simulation is described in Section 10. Concluding remarks are included in Section 11.

2. Introduction

The ever increasing complexity of digital designs makes the verification of the functional correctness of a digital device a challenging endeavor. The verification process is divided into two major phases, design verification (functional testing) and manufacturing verification (fault testing). The technique used for design verification is highly dependent on the design methodology. A graphical representation of the design process with functional testing is included as Figure 2.1. The design process begins by performing a design improvement iteration on a given device. Once the designer completes a design improvement iteration then a model of the device is produced. The device model is then simulated to determine if the design has the desired functional attributes. If the device model passes the functional test then the design process is complete and the model is used to build the designed device. If the device model fails the functional test then another iteration of design improvement and functional testing is performed. The iterative design refinement process continues until the modeled device passes the functional test.

The manufacturing verification process typically employs a well accepted technique referred to as the digital test paradigm. The paradigm consists of four major components: (1) selection of a
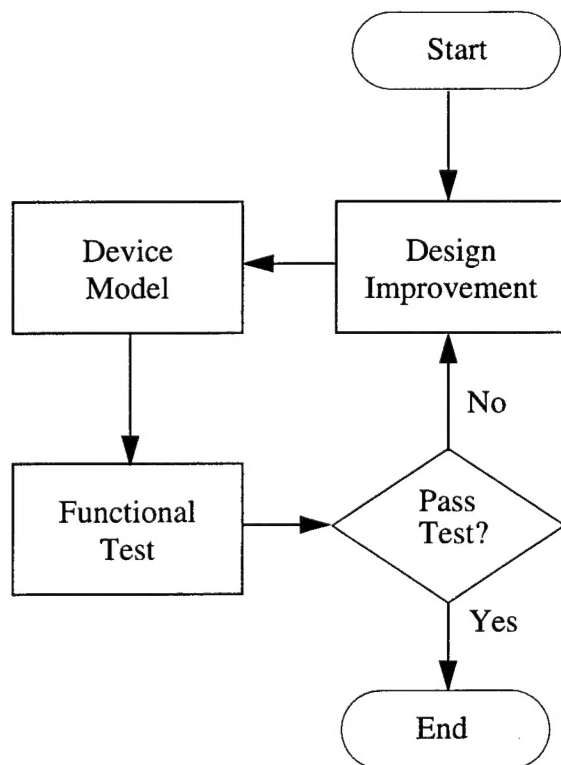
Figure 2.1. Simplified design process diagram showing iterative refinement.

fault model, (2) generation of the fault list of interest, (3) generation of the test patterns to detect the faults contained in the fault list, and (4) estimation of the percentage of faults detected by the application of the test patterns. The four part paradigm, referred to as test construction, is performed after the device is designed but before the device is manufactured. The derived test patterns are then used to determine which manufactured DUTs are faulty.

A block diagram of the test construction process is shown in Figure 2.2. The test construction process begins by applying a fault model to the Device Under Test (DUT) which generates a fault list of relevant faults. Typically, the designer assumes that the fault model is sufficient to represent all manufacturing defects. Thus, applying the fault model to DUT produces a list of all faults which are assumed to completely represent the set of possible manufacturing defects. A Test Pattern Generation (TPG) process is performed next. There are three common categories of TPG: (1) Deterministic Automatic TPG (DATPG), (2) Random Automatic TPG (RATPG), and (3) Manual TPG (MTPG). The purpose of the test patterns is to detect each fault contained in the fault list. Conceptually, if the fault list contains F faults then the fault list can be represented by F faulty DUTs. The
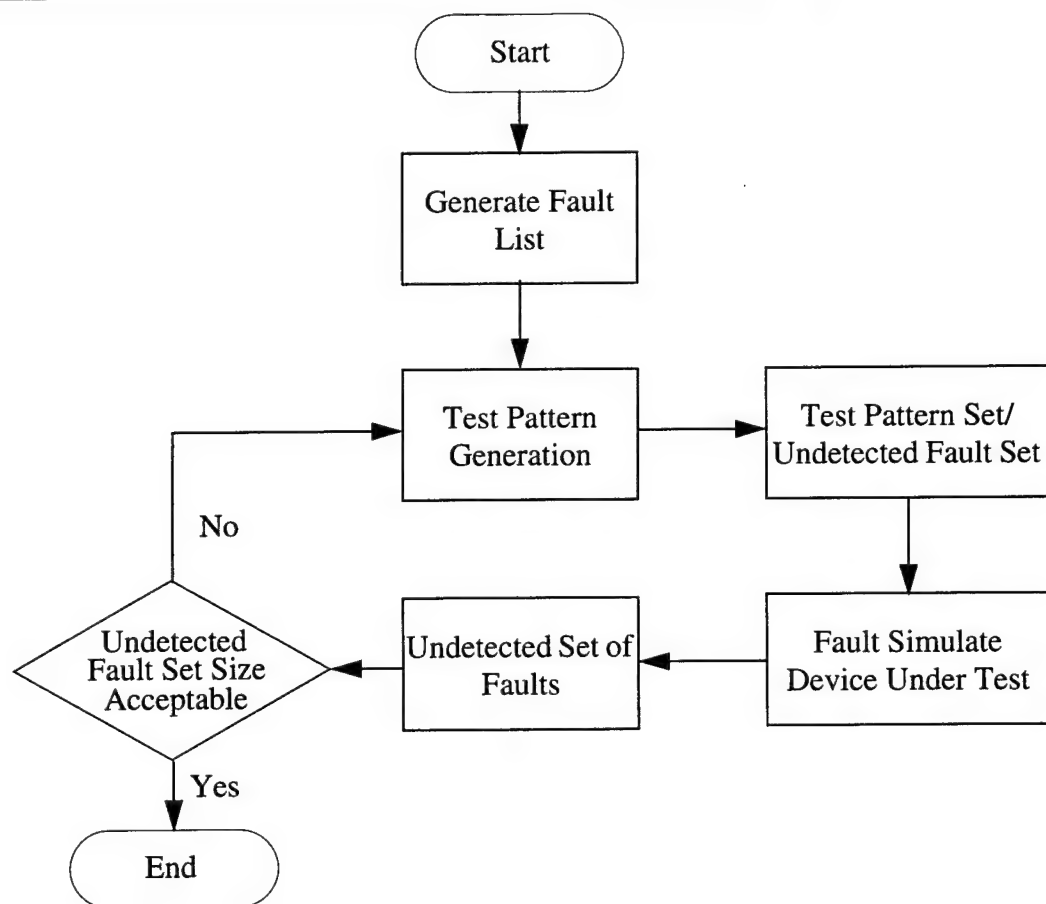
Figure 2.2. Simplified test pattern generation process.

purpose of the test pattern set is to have each of the F faulty DUTs produce one or more output errors. A faulty DUT is detected during testing by comparing the correct output to the DUT's output. If a miscompare occurs then the DUT is faulty.

A fault simulator is used to determine which faults are detected for a given set of test patterns. Conceptually, a fault simulator inserts a fault into the DUT and then the faulty DUT is simulated. If the faulty DUT produces an erroneous output then the fault is detected by the test pattern set. At the end of the fault simulation process a new set of undetected faults is created for the current test pattern set. The designer must then decide if the size of the undetected fault set is acceptable. If the number of undetected faults is acceptable then the test pattern construction process ends, otherwise additional test patterns are produced via TPG and the test pattern evaluation process is repeated. The iterative addition of test patterns and evaluation via fault simulation continues until the number of undetected faults becomes acceptable [1, 174].

The manufacturing test is performed on the manufactured DUT by applying the test pattern set. The output of the manufactured DUT is then compared against the stored correct outputs. If a miscompare occurs then the manufactured DUT is faulty and is repaired or discarded.

The first decision that the designer must make during the test construction process is the selection of a fault model. There are a wide variety of fault models described in literature. A brief description of the accepted faults models is presented next. The purpose of a fault model is to accurately represent the behavior of a faulty component. One possible fault scenario is a fault condition which adds extra delay to the signal propagation through a component. Delay fault models are used to represent faults which cause signal propagation delay [1, 94, 223]. Another common fault scenario is having two input signals shorted together. Bridging fault models are used to represent one signal corrupting another signal [1, 94, 223]. Specifically, a bridging fault occurs when the value of signal A sets the value of signal B. There are two fault models which are commonly used to represent transistor-level fault conditions: (1) stuck-on/off fault model, and (2) stuck-open fault model [1, 94, 223]. For most digital logic architectures a transistor is used as a switching element. The stuck-on/off fault model represents the behavior of transistors which are permanently stuck-on/off. The stuck-open fault model is used to represent a fault condition which is common to CMOS logic. A CMOS circuit has failure modes which cause a signal line to be held at a logic one/ zero value until a parasitic capacitor is discharged/charged. The signal value attached to the parasitic capacitor will eventually discharge/charge and the signal value will assume the correct value. Additionally, functional fault models are employed to evaluated models which are represented at a high-level of abstraction [1, 94, 223]. The major problem with functional fault models is that is difficult to verify that a functional fault model accurately represents low-level physical defects in the DUT. For this reason, functional fault models have found limited acceptance. By far the most

common fault model is the stuck-at fault model [67]. The stuck-at fault model represents a digital signal line being permanently stuck-at a logic one or stuck-at a logic zero value.

Fault simulation is a key component of the test construction process. In general, fault simulators are used to generate data in three areas: (1) to assist in the generation of test patterns, (2) to estimate the test coverage of the DUT, and (3) to generate a fault dictionary [1, 174]. A brief description of the use of fault simulators to assist in test pattern generation was provided previously to explain Figure 2.2. The test coverage of a device is defined to be the probability of detecting a faulty device for a given test pattern set. An estimate for the test coverage is obtained by assuming: (1) the selected fault model is sufficient to represent the failure modes that the DUT can experience, (2) all faults in the fault list are equally likely to occur, and (3) the fault occurrence events associated with faults in the fault list are independent. Using the three assumptions, the expression for the test coverage is derived and is given as

$$C = \frac{n_d}{n_d + n_u} \tag{2.1}$$

where C is the test coverage, n_d is the number of detected faults for a given test pattern set, and n_u is the number of undetected faults for a given test pattern set. The estimation of test coverage is commonly referred to as fault grading. The generation of a fault dictionary is the third category of information which fault simulation can provide. A fault dictionary contains the erroneous response of the DUT for every fault in the fault list for each test pattern in the input set. Typically a fault dictionary is used to locate a fault in the DUT by comparing the erroneous output of the DUT to the erroneous behavior of each fault in the fault dictionary. The use of fault dictionaries is in decline with the increase in the gate count on Integrated Circuits (ICs). Specifically, the size of the fault dictionary for an IC which contains 10^5 gates is extremely large. Also, locating an internal fault on an IC is of little interest since it is typically not possible to repair the located fault.

This report provides an overview of the state-of-the-art techniques used for fault simulation. The primary purpose of this report is to determine the state-of-the-art for fault simulators which are used to estimate the test coverage for the DUT. It is envisioned that the state-of-the-art survey will be used to assist in defining the fault simulation techniques which are applicable to VHDL models. An overview of fault grading techniques and TPG methods is also provided in this report. While fault simulation is the main focus of this report, fault grading and TPG are included to completely describe the test generation, fault simulation, and fault grading process. It is important to realize that fault simulation is a means to assist TPG and estimate fault coverage via fault grading. The desired goal for a tool set is to contain a fault simulation technique which seamlessly augments the TPG process and performs fault grading in an efficient fashion.

This report is organized into ten major sections. Following this introduction, Section 3 provides background concerning fault simulation concepts. A high-level summary of the techniques covered in this report is presented in Section 4. Section 5 provides an overview of uniprocessor based fault simulation techniques. Parallel processor fault simulation techniques are described in Section 6. A review of the use of hardware accelerators to achieve fast fault simulation is presented in Section 7. An overview of existing fault grading techniques is included as Section 8. Likewise, Section 9 provides a brief overview of test pattern generation techniques. An analysis of the applicability of the presented fault simulation methods for use in VHDL based fault simulation is described in Section 10. Concluding remarks are included in Section 11.

3. Fault Simulation Concepts

The first step in this overview process is to describe the structure associated with a generic fault simulation tool. The structure of a generic fault simulation tool is important because it clearly shows what information is required to perform fault simulation and the basic components required for fault simulation. A graphical representation of the generic fault simulation structure is included as Figure 3.1. The information which must be provided by the designer is: (1) the set of test patterns, (2) the fault list, (3) a model of the DUT, and (4) a list of correct outputs for the DUT for the
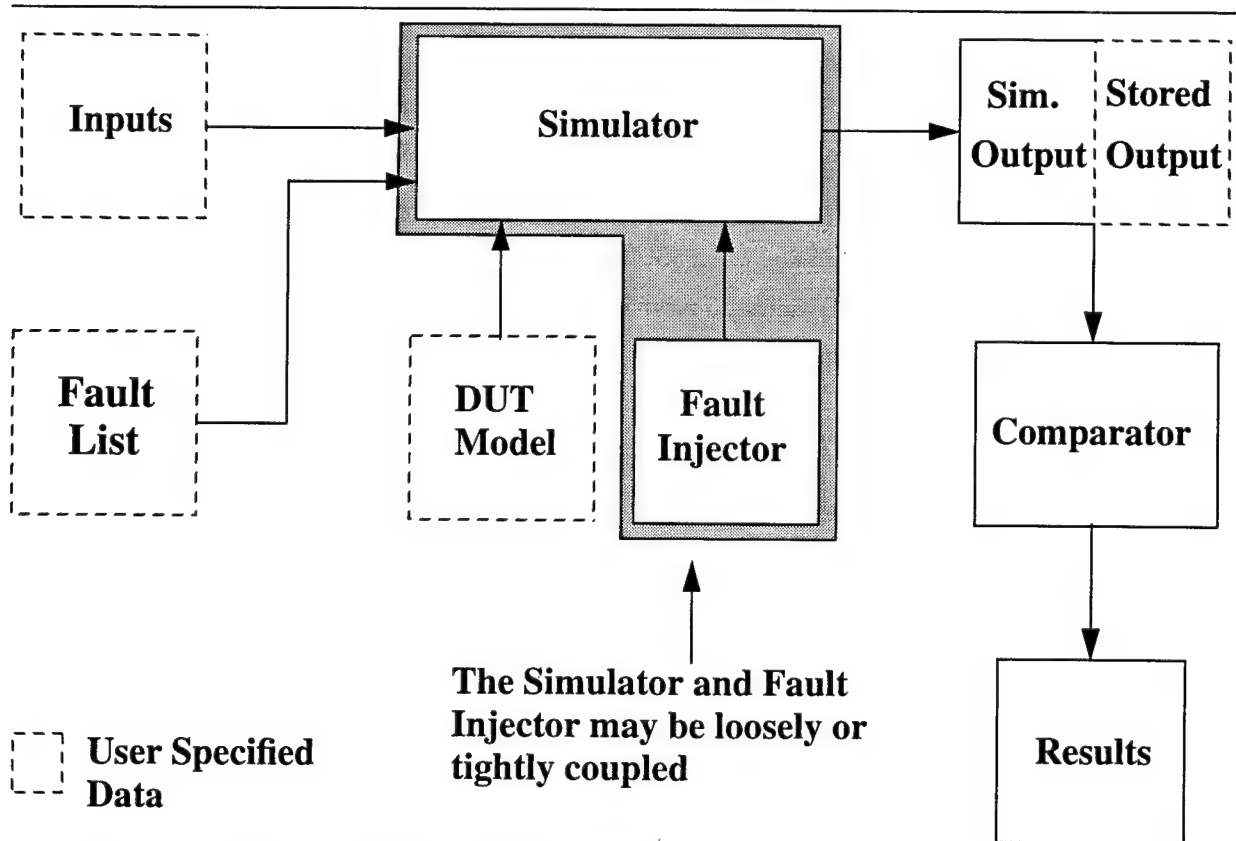
Figure 3.1. General structure of a fault simulator.

set of test patterns. The fault simulation engine reads in the DUT model, a test pattern from the set of test patterns, and then injects one or more faults from the fault list into the DUT. The faulty DUT is simulated, and outputs from the DUT are produced. The simulated outputs of the faulty DUT are compared to the correct stored outputs. The injected fault is detected if the simulated output differs from the stored output. The fault insertion mechanism associated with the fault simulator may be loosely or tightly coupled with the fault simulation engine. An example of a loosely coupled fault insertion method is a technique which uses an existing logic simulator to perform the fault simulation. The fault insertion in this example is achieved by modifying the DUT to incorporate the desired fault. Tightly coupled fault insertion techniques typically require a custom fault simulation engine which can only be used to execute a given fault simulation technique.

For a fault simulator to be useful in assisting the test pattern construction process it must be computationally efficient. Theoretical analysis of a variety of fault simulation techniques indicates that the upper bound associated with the computational complexity of fault simulation is a quadratic function. Specifically, fault simulation requires $O(G^2)$ computational time, where G is the number of components in the device under test [91]. Further insight into the complexity of the fault simulation is obtained by viewing the problem graphically. The fault simulation process can be represented in a three dimensional space with each dimension defined as: (1) the number of components in the device under test given as G, (2) the number of faults in the circuit given as F, and (3) the number of input vectors required to test a device is given as I. The number of faults in the Device Under Test (DUT) is directly proportional to G and is given as

$$F = \alpha(G) \tag{3.1}$$

Likewise, the number of test patterns for a general DUT is also a function of G

$$I = \beta(G) \tag{3.2}$$

The space which must be evaluated by fault simulation is a box shaped volume which is depicted graphically in Figure 3.2. The volume is referred to as a test volume which is proportional to $O(G^3)$ [161]. Various techniques are employed by fault simulation algorithms to reduce the $O(G^3)$ volume to an $O(G^2)$ computational time process [80]. For example, the dropping of detected faults from the fault list after each test pattern is evaluated is one way to reduce the number of fault simulations. The process of deleting detected faults from the fault list is referred to as fault dropping.

All of the aforementioned computational costs assume that the DUT is a combinational circuit which contains no feed back loops. The fault simulation costs associated with evaluating finite state machines are higher than for combinational circuits. Unless otherwise stated throughout this report the computational costs associated with a fault simulation technique are for a combinational circuit. Another factor which effects computational cost is the manner in which components in the
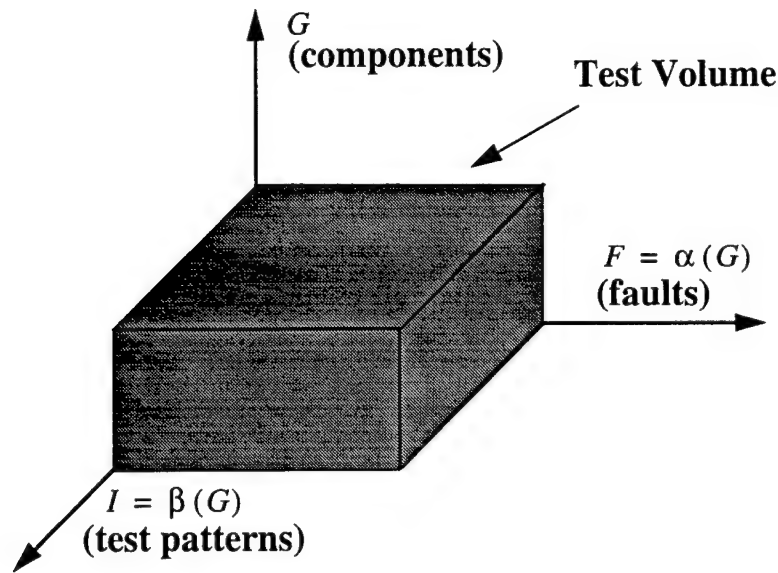
Figure 3.2.   **Space which is evaluated via fault simulation during test coverage estimation.**

DUT are evaluated. The two different generic component evaluation approaches are: (1) compiled simulation and (2) event driven simulation [33]. The first approach consists of evaluating all G gates for each fault simulation and is referred to as compiled parallel fault simulation [24, 33, 57, 107, 118, 225, 226, 235]. An example gate-level circuit is included as Figure 3.3 to assist in describing the compiled fault simulation approach. The first step in evaluating the example DUT for the fault-free case to determine the correct output. A total of five gate evaluations are required for the fault-free simulation. The s-a-0 fault on the output of gate 2 is inserted into the DUT and the faulty circuit is simulated by evaluating all five gates. A compiled approach always evaluates every component in a DUT model for every simulation. Thus, ten gate evaluations are required to simulate the fault free DUT and on faulty DUT for a single input pattern.

The second approach is referred to as event driven simulation. The first stage of event driven simulation is the evaluation of the fault free DUT and storing the signal values of the DUT. A single fault is injected at a signal location in the DUT. The erroneous value caused by the injected fault is propagated through the circuit by evaluating the gates attached to the erroneous signal. Each gate is evaluated in response to an event; that is, the propagation of the erroneous signal. If the output of a gate is erroneous then the gates which utilize the erroneous value as an input are scheduled for evaluation. The simulation of gates is complete when the evaluation list becomes empty. This type of fault simulation is referred to as event driven fault simulation [33, 36, 42, 52, 86, 103, 225, 226, 227]. The evaluation list becomes empty when either the effect of the erroneous signal is masked and the fault is undetectable or the erroneous signal propagates to one or more outputs.
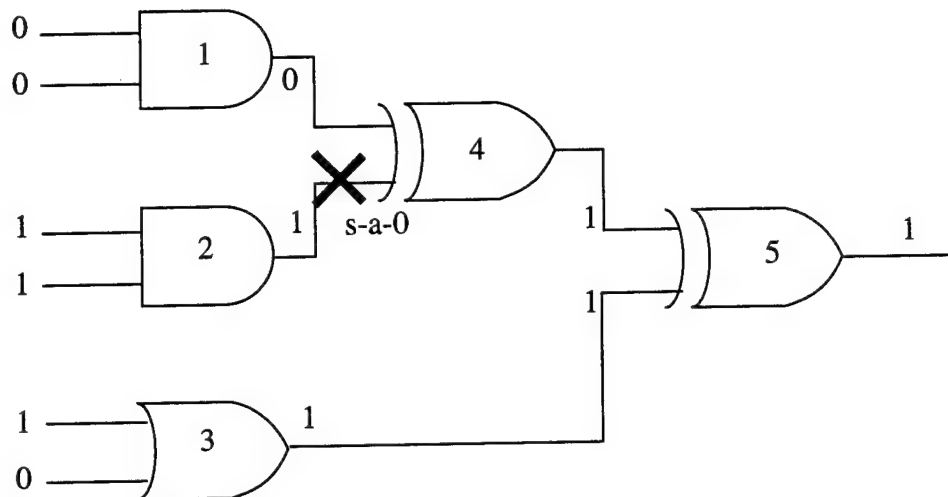
Figure 3.3.   Example fault simulated circuit with a known input pattern.

The example circuit depicted in Figure 3.3 is used to further describe event driven fault simulation. The simulation begins by adding gates 1, 2, and 3 to the evaluation list. As each of the gates is evaluated it is removed from the evaluation list. If the output signal value of an evaluated gate changes then all gates which use the output signal as an input are added to the evaluation list. Once gates 1 and 2 are evaluated gate 4 is added to the evaluation list. Gate 5 is added to the evaluation list once gates 3 and 4 have been evaluated. The fault-free simulation ends when gate 5 is evaluated. The s-a-0 output fault on gate 2 is then inserted. The fault insertion causes gate 4 to be added to the evaluation list. Once gate 4 is evaluated then gate 5 is added to the evaluation list. The fault simulation is one step from completion for the inserted fault when gate 5 is evaluated. Also, for this particular example the inserted fault is detected by the test pattern because the output is erroneous. The last step associated with event driven fault simulation is the restoration of the erroneous signal values to a fault-free value. The restoration is accomplished by copying a saved version of the correct signal values to replace the erroneous signal values. For the example circuit a one value is restored to the output of gates 2, 4 and 5. Once the restoration step is complete then the simulated DUT is ready for another fault simulation. The total number of gate evaluations for the example is seven; that is, five gates are evaluated for the fault free circuit and two gates are evaluated for the faulty circuit.

The primary advantage of event fault simulation is that only the gates which have erroneous inputs are evaluated. Most faults affect only a small percentage of the gates in a DUT, thus event driven fault simulation is more computationally efficient than compiled parallel fault simulation. In fact, event driven fault simulation has a computational cost of $O(G^2)$ [91, 225, 226].

4. Overview of Fault Simulation, TPG, and Fault Grading

The objective of this report is to provide an overview of the state-of-the-art for fault simulation, test pattern generation, and fault grading. Special emphasis is placed on utilizing existing fault simulation techniques for the evaluation of Very High Speed Integrated Circuit Hardware Description Language (VHDL) models. This state-of-the-art survey is used to assist in the development of a fault grading/fault simulation toolset. Specifically, the goal of this research effort is to develop a tool which performs fault grading via fault simulation of a VHDL model using a VHDL compliant simulator. This report surveys three major areas: (1) fault simulation, (2) fault grading, and (3) test pattern generation. A brief overview of the survey associated with each area is described in Sections 4.1, 4.2, and 4.3.

4.1. Fault Simulation Overview

Fault simulation is used to determine the effect of a given fault on a specific Device Under Test (DUT) for a set of input vectors. Typically, the designer specifies a fault model to assist in the fault grading process. The fault model is used to enumerate the set of faults associated with the DUT. If the designer assumes that the fault model is sufficient to represent all relevant DUT faults, then the set of faults derived by applying the fault model to the DUT is complete. There are a variety of fault models for the designer to consider. Delay, bridging, stuck open/stuck closed, and stuck-at are commonly used fault models. The stuck-at fault model is by far the most prevalent. Unless otherwise stated, this overview describes fault simulation techniques which can be used to evaluate stuck-at faults. The survey section of this report expands the discussion to describe extensions of the fault simulation methods presented in the overview which are used for other fault models.

Conceptually, fault simulation is performed by inserting a fault into a DUT and then simulating the faulty DUT for a set of input vectors, commonly referred to as a set of test patterns. Fault simulators have been used for many years to assist designers in developing digital systems. As the complexity of digital systems has increased the demand for fault simulators that handle larger designs has also grown. A time line which details the evolution of fault simulation techniques is included as Figure 4.1. The time line lists the introduction of key fault simulation techniques, the first known date of a published record describing the technique, and the reference which describes the technique. A brief description of each technique is presented in the following paragraphs. A more detailed description of the fault simulation methods is presented in Sections 5 and 6 of this report. The fault simulation techniques are introduced based upon the chronology of development; that is, the fault simulation technique that was developed first is described first.

The most simplistic fault simulation technique is serial fault simulation. The first reported use is presented by Tsiang in 1962 [213]. Serial fault simulation is performed by inserting a single fault in the DUT, and then the faulty DUT is simulated by applying the test pattern set to the faulty DUT.
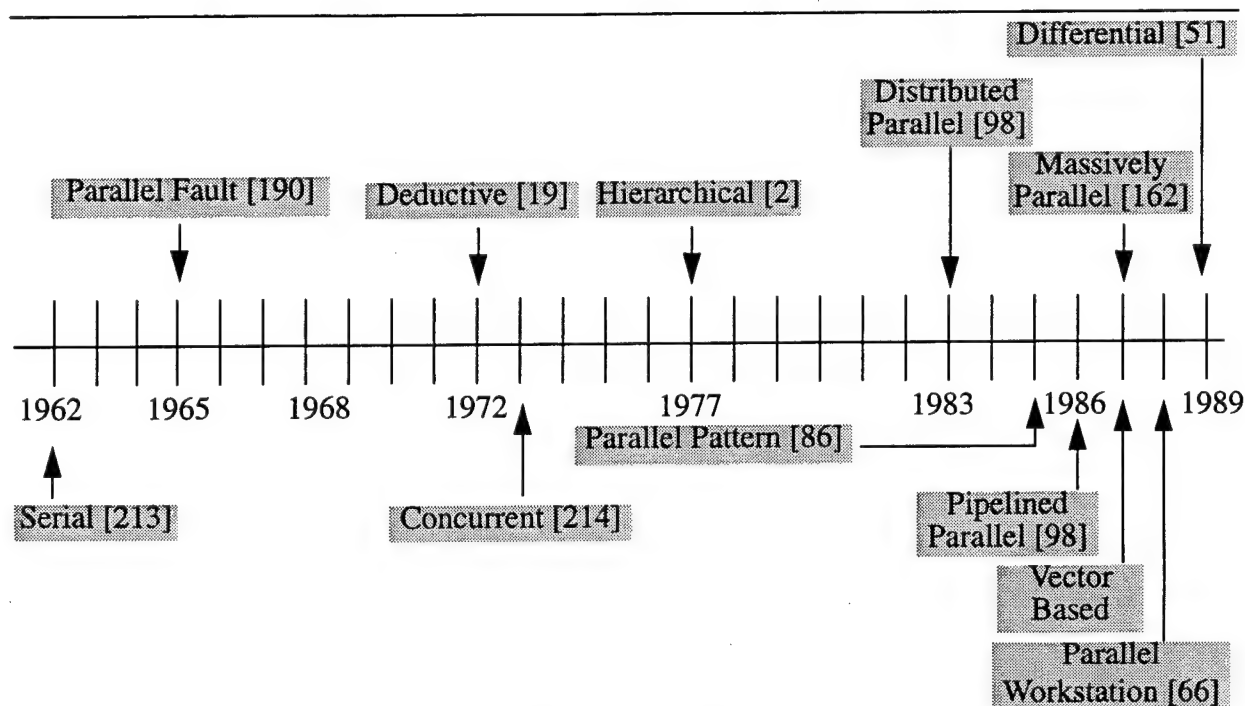
Figure 4.1. Evolutionary time line for fault simulation techniques.

A fault is declared detected if one or more outputs of the simulated faulty DUT do not match the correct DUT outputs. Each fault is evaluated one at a time with the serial fault simulation technique. A list of references where serial fault simulation is utilized is included as Table 4.1. The first column of Table 4.1 lists the reference number, while the second and third columns contain the year of publication and the list of authors associated with the reference. For the references surveyed in this report serial fault simulation is described in papers spanning 1962 [213] to 1995 [64].

The parallel fault simulation technique is an extension of the serial fault simulation method. Serial fault simulation evaluates a single fault for a given input pattern per simulation. Parallel fault simulation evaluates a number of DUTs in parallel per simulation pass. The number of DUTs evaluated in parallel is typically determined by the number of bits in the machine word of the host system which is performing the simulation. For example, if the host machine word width contains W bits, then W DUTs are evaluated in parallel. Specifically, each bit position in the machine word is assigned to a single DUT. Typically, parallel fault simulation is performed for gate-level DUT models. Transforming a gate-level model into a form which is readily simulated via the parallel method is a relatively straightforward process. For example, the evaluation of a two input gate in a DUT model maps to a single host machine instruction. When the instruction is evaluated, all W-bit positions in the input operands are used to calculate the W-bit output operand. Thus, for gate-level models parallel fault simulation is very efficient. Parallel fault simulation is a common method that was utilized in 1965 [190] and is still being used in 1993 [156]. The parallel technique can further be divided into two categories: (1) Parallel Fault (PF) simulation and (2) Parallel Pat-

Table 4.1. Serial fault simulation reference table.

Reference No.	Year Published	Author
[5]	1985	Abramovici, M. and P. R. Menon
[22]	1986	Barzilai, Z., D. K. Beece, L. M. Huisman, V. S. Iyengar, and G. M. Silberman
[43]	1988	Caisso, J.-P. and B. Courtois
[44]	1994	Research Triangle Institute
[49]	1974	Chappell, S. G., C. H. Elmendorf, and L. D. Schmidt
[60]	1984	Davidson, S
[64]	1995[a]	DeLong, T. A, B. W. Johnson, and J. A. Profeta, II
[93]	1982	Hayes, J. P
[110]	1994	Jenn, E., J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson
[143]	1993	Meyer, W. and R. Camposano
[164]	1988	Ozguner, F. and R. Daoud
[180]	1985	Rogers, W. A. and J. A. Abraham
[213]	1962	Tsiang, S. H. and W. Ulrich
[215]	1967	Ulrich, E. G.

a. Submitted for publication date

tern (PP) simulation. The PF technique simulates one good DUT and $W - 1$ DUTs with different faults. Conversely, the PP technique evaluates W input patterns in parallel on one faulty DUT. A list of all references surveyed in this report which utilize parallel fault simulation is included as Table 4.2. The reference number, the year published, the list of authors, and the technique employed (PP or PF) is included in columns one, two, three, and four respectively in Table 4.2.

Deductive fault simulation is a method which can evaluate all faults associated with a DUT in a single simulation pass. The deductive method is typically used only for gate-level models. The deductive technique evaluates the entire set of faults in a symbolic fashion by propagating fault lists through the DUT when a signal in the DUT is updated. A fault on the input of a component is propagated to the output of the component if the input fault causes an erroneous component output. The fault list propagation is performed by using set theory relationships to determine which input faults satisfy the error propagation requirement. The set theory relationships deduce which faults cause erroneous component outputs. A deductive fault simulation begins by adding input faults which produce an error to each input signal's fault list. The components which are attached to the inputs are evaluated next. The simulation proceeds by evaluating components whose inputs are

Table 4.2. Parallel fault simulation reference table.

Reference No.	Year Published	Author	Simulation Method
[1]	1990	Abramovici, M., M. A. Breuer, and A. D. Friedman	PP, PF
[48]	1974	Chang, H. Y.-P., S. G. Chappel, C. H. Elmendorf, and L. D. Schmidt	PF
[49]	1974	Chappell, S. G., C. H. Elmendorf, and L. D. Schmidt	PF
[52]	1990	Cheng, W., and J. H. Patel	PF
[70]	1980	Funatsu, S., M. Takahashi, and M. Shibata	PF
[82]	1984	Goel, P. and P. R. Moorby	PF
[84]	1980	Goel, P., H. Licha, T. E. Rosser, T. J. Stroh, and E. B. Eichelberger	PF
[89]	1966	Hardie, F. H and R. J. Suhocki	PF
[124]	1989	Larrabee, T.	PP
[128]	1980	Levendel, Y. H., and P. R. Menon	PF
[129]	1981	Levendel, Y. H., and P. R. Menon	PF
[142]	1978	Menon, P. R. and S. G. Chappel	PF
[156]	1993	Navabi, S., N. Cooray, and R. Liyanage	PF
[163]	1972	Ozguner, F., W. E. Donath, and C. W. Cha	PF
[180]	1985	Rogers, W. A. and J. A. Abraham	PF
[184]	1984	Saab, I. D. and N. Hajj	PF
[187]	1989	Schulz, M. H., and D. Pellkofer	PP
[190]	1965	Seshu, S.	PF
[201]	1987	Smith, S. and R. von Blucher	PF
[204]	1985	Son, K.	PF
[211]	1975	Thompson, E. W. and S. A. Szygenda	PF
[225]	1985	Waicukauski, J. A., E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy	PP
[226]	1985	Waicukauski, J. A., E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy	PP
[227]	1987	Waicukauski, J. A., V. P. Gupta, and S. T. Patel	PP
[229]	1987	Waicukauski, J. A., E. Lindbloom, B. K. Rosen, and V. S. Iyengar	PP
[235]	1968	Yetter, I. H	PF

known but whose outputs need to be set. The deductive simulation ends when all signal values in the simulation reach a steady-state value. The set of detected faults provided by the simulation is obtained by performing a union of the fault list sets associated with each output signal of the DUT. The list of references surveyed by this report which utilize deductive fault simulation is presented as Table 4.3. The use of the deductive method spans 1972 [19] to 1994 [208].

Table 4.3. Deductive fault simulation reference table.

Reference No.	Year Published	Author
[1]	1990	Abramovici, M., M. A. Breuer, and A. D. Friedman
[19]	1972	Armstrong, D. B
[48]	1974	Chang, H. Y.-P., S. G. Chappel, C. H. Elmendorf, and L. D. Schmidt
[82]	1984	Goel, P. and P. R. Moorby
[84]	1980	Goel, P., H. Licha, T. E. Rosser, T. J. Stroh, and E. B. Eichelberger
[117]	1977	Kjelkerud, E., and O. Thessen
[128]	1980	Levendel, Y. H., and P. R. Menon
[129]	1981	Levendel, Y. H., and P. R. Menon
[142]	1978	Menon, P. R. and S. G. Chappel
[163]	1972	Ozguner, F., W. E. Donath, and C. W. Cha
[174]	1986	Pradhan, D. K.
[180]	1985	Rogers, W. A. and J. A. Abraham
[202]	1988	Smith, S. P. and M. R. Mercer
[208]	1994	Takahashi, N., N. Ishiura, and S. Yajima

The concurrent fault simulation technique is similar to the deductive method. The concurrent technique can evaluate all faults associated with the DUT in a single simulation pass. The faults associated with the DUT are symbolically processed with the concurrent method. Unlike the deductive method which stores the fault lists with each signal, the concurrent method stores the list of faults with each component. The fault list propagation for an unevaluated component begins by gathering the fault lists propagated by the components attached to the input signals of the uneval-uated component. The fault list associated with each input to a component is evaluated by applying the fault condition to the component input and evaluating the component. If the output of the com-ponent is erroneous then the fault is added to the component's fault list. The concurrent technique begins by evaluating the components attached to the inputs of the DUT which are referred to as

input components. The fault list propagation associated with an input component is performed by evaluating component input faults which cause input errors. Each input fault which causes a component output error is stored in the input component's fault list. The concurrent technique continues by evaluating components whose inputs are known but whose outputs are yet to be evaluated. The concurrent technique ends when all signals in the DUT reach a steady-state value. The set of faults detected by the concurrent simulation is determined by performing a union of all the fault sets associated with the outputs of the DUT.

One of the primary benefits of the concurrent technique is that it can be applied to a model at any level of abstraction. In fact, the concurrent method can be used to evaluate DUT models which contain components at various levels of abstraction. The use of multiple levels of abstraction is the primary reason for the popularity of the concurrent method. The list of concurrent fault simulation references surveyed for this report is given as Table 4.4. Concurrent fault simulation was first introduced by Ulrich and Baker in 1973 [216] and is available today in the form of several commercial fault simulation tools offered by Attest [20], Ikos systems [104], and ZyCAD [236].

Hierarchical fault simulation is a method which exploits design hierarchy to speed the fault evaluation process. The design hierarchy is exploited by having each component in the DUT represented at the highest level of abstraction, except the component which is at fault. The component which is being evaluated via fault simulation is represented by a low-level model, such as a gate-level model. Thus, hierarchical fault simulation creates a unique model of the DUT for each component which is evaluated via fault simulation. Once the evaluation of a given component is completed then the fault simulator selects another unevaluated component for simulation and loads the appropriate hierarchical DUT model. A DUT model which contains design hierarchy requires less computational resources to simulate than a gate or transistor level DUT model. The more high-level components that a DUT contains the better the increase in efficiency obtained by hierarchical fault simulation.

The simulation technique employed to evaluate the hierarchical model varies. Both serial and concurrent fault simulation methods have been used to evaluate hierarchical models. Hierarchical fault simulation has been in use since 1977 [2] and has been reported in the literature as recently as 1995 [99]. The list of references surveyed by this report which employ hierarchical simulation is included as Table 4.5.

Serial, deductive, concurrent, and hierarchical fault simulation methods have one attribute in common; that is, the techniques are designed to execute on a uniprocessor host computer. One obvious way to increase speed during the solution of a computationally intensive problem is to use parallel processor architectures to increase the amount of computational resources. The parallel processing techniques are organized in this survey by parallel processor hardware architecture. The architecture of a parallel machine has a large impact on the types of problems that a given machine

Table 4.4. Concurrent fault simulation reference table.

Reference No.	Year Published	Author
[1]	1990	Abramovici, M., M. A. Breuer, and A. D. Friedman
[2]	1977	Abramovici, M., M. A. Breuer, and K. Kumar
[20]	1995	Attest Software, Inc.
[31]	1992	Bose, S. and P. Agrawal
[32]	1982	Bose, A., P. Kozak, C.-Y. Lo, H. N. Nham, E. Pacas-Skewes, and K. Wu
[39]	1983	Bryant, R. E. and M. D. Schuster
[40]	1985	Bryant, R. E. and M. D. Shuster
[46]	1986	Chan, T. and E. Law
[55]	1980	d'Abreu, M. A., and E. W. Thompsond'Abreu, M. A., and E. W. Thompson
[82]	1984	Goel, P. and P. R. Moorby
[95]	1980	Henckels, L. P., K. M. Brown, and C. Lo
[104]	1995	Ikos Systems, Inc.
[125]	1992	Lee, D. H. and S. M. Reddy
[128]	1980	Levendel, Y. H., and P. R. Menon
[129]	1981	Levendel, Y. H., and P. R. Menon
[132]	1987	Lo C. Y., H. N. Nham, and A. K. Bose
[174]	1986	Pradhan, D. K
[180]	1985	Rogers, W. A. and J. A. Abraham
[186]	1977	Schuler, D. M and R. K. Cleghorn
[189]	1984	Schuster, M. D. and R. E. Bryant
[195]	1985	Shih, H. C, J. T. Rahmeh, and J. A. Abraham
[196]	1986	Shih, H. C, J. T. Rahmeh, and J. A. Abraham
[200]	1984	Smith, L. T. and R. R. Rezac
[207]	1995	Synopsis
[214]	1985	Ulrich, E
[216]	1973	Ulrich, E. G. and T. Baker
[217]	1980	Ulrich, E, D. Lacy, N. Phillips, J. Tellier, M. Kearney, T. Elkind, and R. Beaven
[236]	1995	ZyCAD

Table 4.5. Hierarchical fault simulation reference table.

Reference No.	Year Published	Author
[2]	1977	Abramovici, M., M. A. Breuer, and K. Kumar
[47]	1987	Chang, H. P. and J. A. Abraham
[61]	1986	Davidson, S. and J. L. Lewandowski
[72]	1988	Gai, S., P. L. Montessoro, and F. Somenzi
[73]	1988	Gai, S., P. L. Montessoro, and F. Somenzi
[74]	1986	Gai, S., F. Somenzi, and E. Ulrich
[87]	1986	Guzolek, J. F., W. A. Rogers, and J. A. Abraham
[99]	1995	Hsiao, M. S. and J. H. Patel
[103]	1990	Hwang, T.-S., C. L. Lee, W. Z. Shen, and C. P. Wu
[137]	1988	Machlin, D., D. Gross, S. Kadkade, and E. Ulrich
[144]	1993	Meyer, W. and R. Camposano
[147]	1991	Montessoro, P. L. and S. Gai
[148]	1988	Motohara, A. M. Murakami, M. Urano, Y. Masuda, and M. Sugano
[159]	1988	Nicholls, W. H. and M. Soma
[178]	1987	Rogers, W. A., J. F. Guzolek, and J. A. Abraham
[179]	1985	Rogers, W. A. and J. A. Abraham
[180]	1985	Rogers, W. A. and J. A. Abraham
[185]	1979	Schuler, D. M., T. E. Baker, R. S. Fisher, SS. Hirshhorn, M. B. Hommel, H. J. McGinness, and R.V. Bosslet

is well suited to solve. For this reason more insight is to be gained by studying the parallelization techniques grouped by hardware architecture.

The first reported parallel processor architecture employed for fault simulation is a distributed parallel processor. A parallel processor architecture is considered distributed if there is a large communication time penalty associated with interprocessor communication. The distributed parallel fault simulation techniques surveyed in this report center on partitioning the DUT into regions which can be evaluated independently. The independent regions are then assigned to independent processors. The number of independent regions into which a DUT can be partitioned is highly dependent on the structure of the DUT. The first published distributed parallel processing method is written by Goel in 1981 [76]. Conversely, Huisman et al has done the most recent work in this

area; that is 1990 [100- 102]. A listing of the references surveyed relating to parallel distributed fault simulation is included as Table 4.6.

Table 4.6. Distributed parallel fault simulation reference table.

Reference No.	Year Published	Author
[76]	1981	Goel, P
[100]	1990	Huisman, L. M. and R. Daoud
[101]	1990	Huisman, L., I. Nair, and R. Daoud
[102]	1990	Huisman, L., I. Nair, and R. Daoud
[130]	1983	Levendel, Y. H, P. R. Menon, and S. H. Patel
[150]	1989	Mueller-Thuns, R. B., D. G. Saab, R. F. Damiano, and J. A. Abraham

A pipelined processing architecture is another type employed to perform fault simulation. A pipelined architecture is optimized to perform a sequential algorithm by assigning a processing element to each step in the algorithm. As the sequential algorithm executes, data is passed down the array of processing elements with the last processing element producing the output of the sequential algorithm. The pipeline approach is similar to the instruction pipelining which is employed on modern 32-bit microprocessors to increase performance.

The first step in translating a sequential algorithm to a pipelined architecture is dividing the sequential algorithm into partitions which require approximately equal amounts of computational resources. The requirement of equal resources assures that one element of the pipeline does not cause the entire pipeline to stall. For pipelining to be efficient, the sequential algorithm has to have sufficient complexity to support partitioning. Both the aforementioned deductive and concurrent methods have sufficient algorithmic complexity to support pipelining. The list of references which employ pipelining is included as Table 4.7.

Table 4.7. Pipelined parallel fault simulation reference table.

Reference No.	Year Published	Author
[8]	1989	Agrawal, P., V. D. Agrawal, K.-T. Cheng, and R. Tutundjian
[31]	1992	Bose, S. and P. Agrawal
[131]	1995	Li., Y.-L. and C.-W. Wu
[165]	1988	Ozguner, F., C. Aykanat, and O. Khalid
[206]	1986	Stein, A. J., D. G. Saab, and I. N. Hajj

Vector-based parallel processor architectures are yet another type of parallel machine which is used to perform fault simulation. Vector-based machines are optimized to perform vector operations such as matrix multiplication at a high level of throughput. Supercomputers typically employ a vector architecture to enhance the evaluation of computationally intensive scientific/engineering algorithms. Parallel fault simulation is ideally suited to execute on a vector-based machine. Specifically, each component in the DUT is evaluated as W parallel machines using W-bit wide vector operations. Ishiura et al [106] in 1985 is the first known reported use of a vector based architecture to perform fault simulation. The list of references surveyed by this report which use vector based architectures is included as Table 4.8.

Table 4.8. Vector-based fault simulation reference table.

Reference No.	Year Published	Author
[25]	1992	Bataineh, A., F. Ozguner, and I. Szauter
[59]	1989	Daoud, R. and F. Ozguner
[105]	1990	Ishiura, N., M. Ito, and S. Yajima
[106]	1987	Ishiura, N., M. Ito, and S. Yajima
[152]	1994	Nagumo, T., M. Nagai, T. Nishida, M. Miyoshi, and S. Miyamoto

Massively parallel architectures have also been employed to perform fault simulation. A parallel architecture which contains more than 100 processing elements is defined to be a massively parallel machine. The connection machine which can contain up to 64K processors is a good example of a massively parallel machine. The serial and parallel fault simulation methods are techniques which are easy to implement on a massively parallel architecture. Parallelization of uniprocessor fault simulation algorithms is achieved in one of two ways. One method is to assign each component in the DUT to a processor. With the processor-per-component technique the number of steps required to evaluate the DUT is now a function of the number of component levels in the DUT and not the size of the DUT. The second approach is to divide the fault list associated with the DUT into n equal partitions. Each fault list partition is then evaluated with a separate processor which is executing a uniprocessor fault simulation technique. The list of references which are surveyed by this report and use a massively parallel machine is presented as Table 4.9.

Another parallel architecture that can be used to perform fault simulation is a group of workstations which are interconnected by a common network. The main benefit of this approach is that most engineering design centers contain a number of networked workstations. Thus, fault simulation on parallel workstations can be performed without having to buy custom parallel processing

Table 4.9. Massively parallel fault simulation reference table.

Reference No.	Year Published	Author
[153]	1989	Narayanan, V. and V. Pitchumani
[154]	1988	Narayanan, V. and V. Pitchumani
[155]	1992	Narayanan, V. and V. Pitchumani
[162]	1987	Ostapko, D. L., Z. Barzilai, and G. M. Silberman

hardware. Also, the fault simulations can be run when the group of workstations is idle such as at night and on weekends.

Fault simulation on parallel workstations is typically performed by partitioning the fault list associated with the DUT into n equal partitions. Each fault list partition is then evaluated on a separate workstation using a uniprocessor fault simulation technique. Table 4.10 lists the references surveyed by this report which use parallel workstation fault simulation techniques.

Table 4.10. Parallel workstation fault simulation reference table.

Reference No.	Year Published	Author
[66]	1988	Duba, P. A., R. K. Roy, J. A. Abraham, and W. A. Rogers
[139]	1990	Markas, T., M. Royals, and N. Kanopoulos

The newest fault simulation technique surveyed by this report is differential fault simulation. The differential fault simulation method is primarily used to evaluate sequential gate-level models. The approach taken with differential fault simulation is to convert the sequential gate-level model into a combinational problem. The conversion occurs by adding controllability and observability to the state elements of the sequential model. The controllability feature allows the setting of the state of the DUT to any value, while the observability allows the reading of the current state value. The simulation technique begins by initializing the state of the DUT to some predefined value. The first test pattern is applied, and the DUT is simulated. The fault-free output and fault-free next state values are stored. The first fault is inserted into the DUT, and the faulty DUT is simulated. If the faulty DUT produces an output error then the fault is marked as detected, else the next state of the faulty DUT is stored. The evaluation of faults is continued until all of the undetected faults are evaluated. The next test pattern is then applied to the fault-free DUT. The fault-free state of the DUT is restored and the circuit is simulated. The fault-free output and next state are then stored. The first undetected fault is then evaluated by restoring the stored state of the faulty DUT. If the fault is undetected, then the next state of the DUT is saved. The evaluation of undetected faults is continued until all undetected faults are simulated. The differential method continues with the next test

pattern. The differential fault simulation ends when either all of the faults are detected or all of the test patterns have been evaluated. The references associated with the differential method are included as Table 4.11.

Table 4.11. Differential fault simulation reference table.

Reference No.	Year Published	Author
[51]	1989	Cheng, W.-T. and M.-L. Yu
[220]	1991	Vandris, E., and G. Sobelman
[221]	1990	Vandris, E., and G. Sobelman

4.2. Fault Grading Overview

Fault simulation of the DUT provides information concerning the quality of the test pattern set for a given DUT. Specifically, fault simulation of the entire fault set for a given test pattern set determines which faults are detected by the test pattern set and which are not detected. The fault detection data is then used to determine the percentage of detected faults associated with the DUT when the test pattern set is applied to the DUT. The percentage of detected faults is referred to as fault coverage and the process of estimating fault coverage is referred to as fault grading. A graphical representation of the categories of existing fault grading methods is depicted as Figure 4.2. This section provides an overview of existing fault grading techniques which are surveyed by this report. The low-level details associated with the various fault grading techniques are presented in Section 8.

Fault grading techniques can be subdivided into three major categories: (1) exhaustive methods, (2) statistical methods, and (3) probabilistic methods. Exhaustive methods evaluate all mod-

Fault Grading Techniques

Exhaustive   Statistical   Probabilistic

Single Sided Confidence Interval   Double Sided Confidence Interval
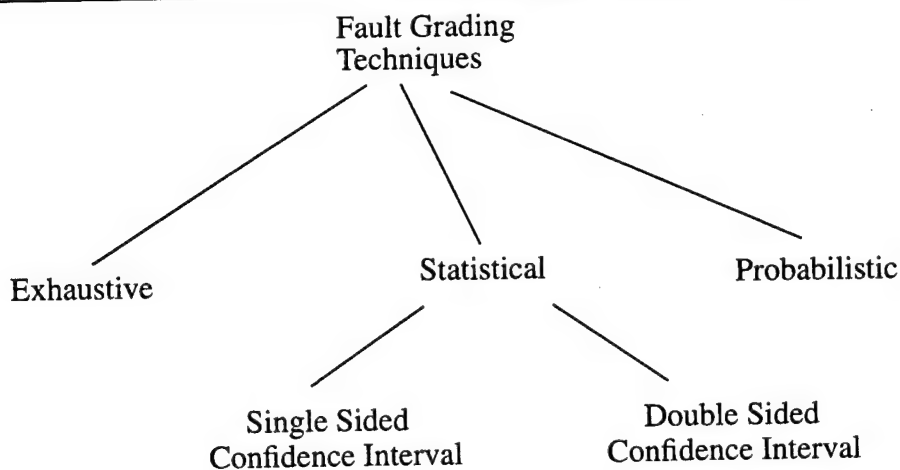
Figure 4.2. Fault grading techniques organized by category.

eled faults associated with the DUT via fault simulation to perform fault grading. Specifically, the exhaustive approach estimates fault coverage by evaluating all faults in the list of faults via fault simulation. Conversely, the statistical approach entails randomly sampling the list of faults, evaluating the sample set of faults using fault simulation, and calculating a point estimate for fault coverage. A confidence interval is typically calculated to quantify the accuracy of the point estimate. One of two techniques is commonly employed to calculate the confidence interval of the point estimate: (1) a double sided confidence interval is employed, and (2) a single sided confidence interval is used. Typically the lower side of the confidence interval is used as the reported coverage estimate. The value of the lower-side confidence interval is always less than the point estimate, and as such, always provides a conservative estimate of the actual fault coverage.

The Department of Defense (DoD) requires that fault grading be performed using a specific methodology; that is, MIL-STD-883 defines the fault grading methodology. Both exhaustive and statistical techniques are allowed under MIL-STD-883. Specifically, when a statistical fault grading approach is used then the lower side of a double sided confidence interval is calculated.

Probabilistic methods form the third fault grading category. The probabilistic fault grading methods are based on measuring the observability and controllability of signals in the DUT for a given set of test patterns. The controllability/observability values are then used to provide a qualitative estimate of the fault coverage. The main benefit of probabilistic methods is that they are computationally inexpensive to perform. Due to the low computational cost, probabilistic methods can provide a fault coverage estimate in a fraction of the amount of time required to perform fault grading using either traditional methods/MIL-STD-883. The main deficiency with probabilistic fault grading methods is that is difficult, if not impossible, to determine the accuracy of the fault coverage estimate. The only known way to determine the accuracy of the probabilistic fault coverage estimate is to perform traditional/MIL-STD-883 fault grading to estimate fault coverage. The probabilistic fault coverage estimate is then compared with the fault simulated coverage estimate to measure the accuracy of the approximate estimate. Having to perform fault simulation coverage estimation to determine the accuracy of probabilistic fault grading techniques mitigates the computational cost savings associated with approximate methods. The inability to determine the accuracy of probabilistic methods has limited their use by designers.

4.3. Test Pattern Generation Overview

The generation of the test pattern set which is used to evaluate the DUT is referred to as Test Pattern Generation (TPG). There are numerous TPG techniques, however, the techniques can be grouped into three major categories: (1) Deterministic Automatic Test Pattern Generation (DATPG), (2) Random Automatic Test Pattern Generation (RATPG), and (3) Manual Test Pattern Generation (MTPG). A graphical depiction of the three major TPG categories and several subcat-

egories is included as Figure 4.3. An overview of each TPG category depicted in Figure 4.3 is provided in the following paragraphs, while the low-level details are presented in Section 9.

RATPG is a technique where probabilities are used to select the next test pattern. The selection process is performed using a sampling with or without replacement strategy. Specifically, for the sampling without replacement case once a test pattern is selected it is removed from the set of available (unselected) test patterns [63]. The selected fault is then fault simulated to determine which currently undetected faults are detected with the new test pattern. If no undetected faults are detected, then the new test pattern is deleted. Conversely, if faults are detected, then the new test pattern is added to the test pattern set and fault grading is performed. The fault grading process is employed to determine the fault coverage estimate for the new test pattern set. If the coverage estimate meets or exceeds the specified fault coverage value, then the RATPG process ends, otherwise the next randomly generated test pattern is selected and the evaluation process is repeated.

The sampling technique employed for RATPG can be divided into two categories: (1) uniform selection, and (2) weighted selection. The uniform selection method assumes that all available test patterns are equally likely and are sampled using a uniform probability density function. The weighted selection process modifies the uniform probability density function by analyzing the structure of the DUT. The objective of the weighted selection method is to increase the likelihood of selecting an input pattern which detects a random pattern resistant fault. A fault is considered to be random pattern resistant if there are very few input patterns which detect the fault. Thus, weighted selection modifies the selection probability density function to maximize the probability of selecting an input pattern which detects one or more faults. A list of references which utilize RATPG is included as Table 4.12.
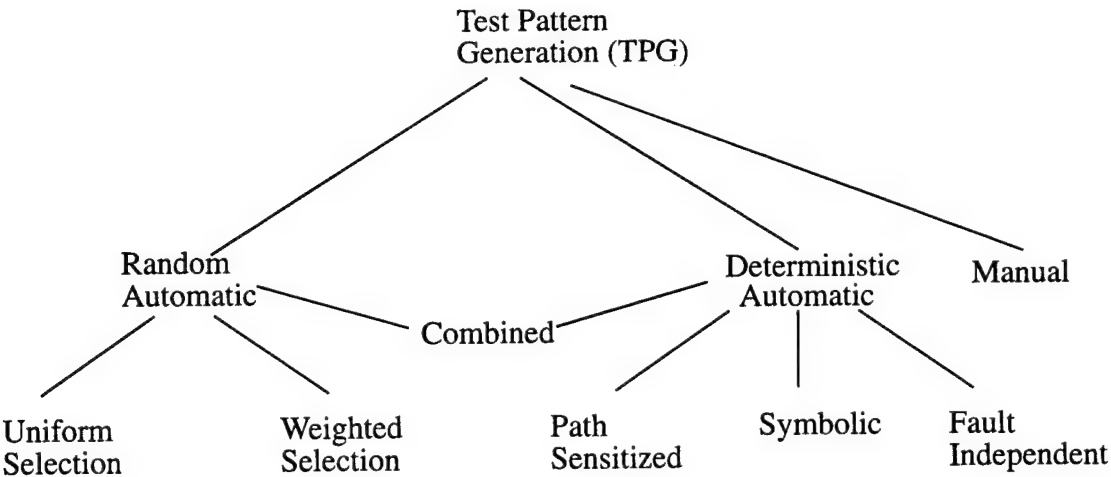
Figure 4.3.   Types of test pattern generation techniques.

Table 4.12.   Random automatic test pattern generation reference table.

Reference No.	Year Published	Author
[35]	1985	Brglez, F., P. Pownall, and R. Hum
[41]	1982	Carter, J. L., S. F. Dennis, V. S. Iyengar, and B. K. Rosen
[63]	1986	Debany, W. H., P. K. Varshney, and C. R. P. Hartman
[124]	1989	Larrabee, T.
[192]	1989	Seth, S. C., V. D. Agrawal, and H. Farhat
[228]	1989	Waicukauski, J. A., E. Lindbloom, E. B. Eichelberger, and O. P. Forlenza

The primary difference between DATPG and RATPG methods is that DATPG techniques use an algorithm which evaluates the internal structure of the DUT to determine a test pattern which detects a specific fault. Some DATPG techniques also provide information on other faults which are detected by the generated test pattern. For example, the D-algorithm uses fault collapsing to identify additional faults which are detected with a given input pattern [181]. In general fault simulation is required to determine the complete set of faults detected by an DATPG produced test pattern. DATPG methods can be delineated into three categories: (1) path sensitized methods, (2) symbolic, and (3) fault independent techniques.

The path sensitized category contains the majority of DATPG algorithms. The common attribute associated with the path sensitized approach is that an algorithm searches for a path through the DUT which activates a given fault and propagates the error associated with the activated fault to one or more outputs of the DUT. Path sensitized methods search to find an input pattern that provides the desired fault activation and output error. Typically, the input pattern search process is exhaustive in nature with path sensitized techniques. Specifically, most path sensitized algorithms are guaranteed to find an input pattern which detects a given fault if the fault can be detected. For a complex DUT, the exhaustive search process can take an inordinate amount of time. Some path sensitization algorithms allow the designer to specify a maximum search space limit to bound the amount of time spent looking for a test pattern. The list of path sensitized DATPG methods is included as Table 4.13.

The symbolic DATPG category determines test patterns for gate-level designs by manipulating equations. A symbolic technique begins by converting a gate-level model to a set of Boolean expressions. The Boolean expressions are then manipulated using a predefined sequence of operations to produce a final set of equations. Each resulting equation represents an input condition which when satisfied detects a given fault. Specifically, an input which causes a resulting equation to be a logical one produces a test pattern which detects a given fault. The primary limitation of

Table 4.13. Path sensitized automatic test pattern generation reference table.

Reference No.	Year Published	Author
[14]	1976	Akers, S. B.
[45]	1978	Cha, C. W., W. E. Donath, and F. Ozguner
[69]	1983	Fujiwara, H. and T. Shimono
[79]	1981	Goel, P.
[83]	1981	Goel, P. and B. C. Rosales
[114]	1987	Kirkland, T., and M. R. Mercer
[151]	1976	Muth, P.
[181]	1966	Roth, J. P.
[182]	1967	Roth, J. P., W. G. Bouricius, and P. R. Schneider
[188]	1988	Schulz, M. H., E. Trischler, and T. M. Sarfert

symbolic DATPG methods is that the number of symbols required to represent any nontrivial DUT is quite large. Thus, the manipulation of the DUT equations is quite involved and for this reason symbolic DATPG is rarely used. A list of references which discuss symbolic TPG is presented as Table 4.14

Table 4.14. Symbolic automatic test pattern generation reference table.

Reference No.	Year Published	Author
[71]	1986	Gaede, R. K., M. R. Mercer, K. M. Butler, and D. E. Ross
[119]	1978	Kohavi, Z.
[123]	1992	Kung, C. and C. Lin
[140]	1986	McCluskey, E. J.
[174]	1986	Pradhan, D. K.

The final DATPG category is fault independent methods. The key feature of fault independent methods is that the algorithm is not attempting to detect a specific fault but to determine a large fault set detected by the generated test pattern. Most fault independent methods require a gate-level model and begin by selecting an output value for the DUT. Each output value of the DUT is selected to satisfy a path sensitization criteria. Specifically, the gate which sets the output value must have a controlling input. An input is considered to be controlling if changing the input causes the output of the component to change value. For example, a two input AND gate with a 10 input pattern has the 0 input as a controlling input. The fault independent methods search for input com-

binations on gates which cause long sensitized paths. The sensitized path is determined by working backwards through the DUT and noting the continuous chain of controlling inputs. Changing the value of any of the controlling inputs on the continuous chain causes a DUT output error. The list of fault independent DATPG references is included as Table 4.15.

Table 4.15. **Fault independent automatic test pattern generation reference table.**

Reference No.	Year Published	Author
[13]	1979	Airapetian, A. N, and J. F. McDonald
[28]	1983	Benmehrez, C., and J. F. McDonald
[209]	1971	Thomas, J. J.
[230]	1975	Wang, D. T.

The combined TPG technique attempts to exploit the best attributes associated with RATPG and DATPG. The combined technique begins by performing RATPG. The RATPG process is continued until the rate at which faults are detected from the undetected fault set drops below a predefined threshold. The rate at which faults are detected from the undetected fault set slows as test patterns are added to the test pattern set with RATPG because the initial test patterns locate the easy to detect faults. Eventually the undetected fault set comprises only random pattern resistant faults. The idea with combined TPG is to have an DATPG technique determine test patterns for the remaining set of random pattern resistant faults. Thus, RATPG is used to generate test patterns to detect the set of easy to detect faults. RATPG is far more efficient than DATPG techniques at detecting nonrandom pattern resistant faults (easy to detect faults). Using RATPG to generate test patterns for the set of easy to detect faults decreases the computational resources required for TPG. Conversely, RATPG tends to become inefficient when used to detect random pattern resistant faults. DATPG techniques can locate a test pattern to detect a random pattern resistant fault more efficiently than RATPG. For these reasons, the combined technique exploits the best attributes of both RATPG and DATPG. A list of references which utilize combined TPG is presented as Table 4.16.

Table 4.16. **Combined automatic test pattern generation reference table.**

Reference No.	Year Published	Author
[4]	1986	Abramovici, M., J.J. Kulikowski, P. R. Menon, and D. T. Miller
[35]	1985	Brglez, F., P. Pownall, and R. Hum
[124]	1989	Larrabee, T.

The distinguishing characteristic of the MTPG technique is that the test patterns are selected manually by the designer. Typically, the first test patterns selected are also used by the designer to verify that the DUT has the correct functional mapping. Like RATPG, each new test pattern is evaluated via fault simulation to determine if any faults from the undetected fault set are detected. If no faults are detected by the new test pattern then the test pattern is discarded, otherwise the new test pattern is added to the test pattern set. Fault grading is then employed to determine if the specified level of fault coverage is achieved. If the test pattern set achieves or exceeds the fault coverage specification then the MTPG process is ended, else the MTPG process is continued. Subsequent test patterns are selected after the designer performs some analysis of the DUT. After each test pattern is selected a fault simulation is performed to determine if any undetected faults are detected. If no additional faults are detected then the new test pattern is discarded, else the new test pattern is added to the set of test patterns. The manual selection process continues until the number of undetected faults reaches an acceptable level.

5. Uniprocessor Fault Simulation

Fault simulation techniques have been implemented using a wide variety of diverse computer architectures. Typically, fault simulation techniques are first developed on uniprocessor architectures and then extended/modified to exploit other architectures such as parallel or pipelined. Thus, a thorough understanding of uniprocessor fault simulation techniques provides the necessary foundation required to understand fault simulation techniques for other architectures. There are a number of fault simulation techniques based on the use of a uniprocessor. As with any technology, improvements in fault simulation have occurred over a large span of time as more stringent performance requirements surfaced. An overview of the specific uniprocessor fault simulation techniques surveyed by this report are presented in the following subsections. Each subsection describes one fault simulation method.

5.1. Serial Fault Simulation

The most fundamental fault simulation approach is referred to as serial fault simulation. The purpose of describing the serial fault simulation technique is to provide insight into the limitations of using the most straightforward solution to fault simulation. Also, a greater appreciation for the power of more complex fault simulation techniques is gained by understanding the limitations of serial fault simulation.

Serial fault simulation consists of injecting one fault at a time in the DUT and simulating the faulty DUT for all input patterns [5, 22, 43, 49, 60, 93, 143, 164, 180, 213, 215]. With this approach, each fault simulation requires that G gates be evaluated for each input vector. If the fault is detected by the ith input vector then the fault simulation for the current fault can be stopped.

Stopping a fault simulation in this fashion is referred to as fault dropping. After the fault is detected or all I input patterns are evaluated the detection status of the fault is recorded and the fault is removed from the fault list. The next fault is selected, and the fault evaluation process is performed again. A high-level description of the serial fault simulation algorithm is included as Figure 5.1 to further illustrate the steps associated with serial fault simulation. The primary drawback to this approach is that G gates are evaluated for every input pattern applied during fault simulation even though the injected fault typically only affects only a small subset of G. The other important point to remember is that a fault-free simulation run is performed to generate the fault-free outputs of the DUT. The computational cost associated with the serial fault simulation technique for a single fault is

$$c_{s_i} \propto GI_i \tag{5.1}$$

where c_{s_i} is the computational cost of the serial fault simulation for the ith fault pattern, and I_i is the number of input patterns required to fault simulate with fault dropping the ith fault. If fault dropping is employed then the number of test vectors per fault is between 1 and I. If fault dropping is not employed then $I_i = I \forall \{i \in F\}$. The total computational cost for the serial fault simulation technique is obtained by summing c_{s_i} over all faults. Performing this summation provides

$$\kappa_b \propto \sum_{i=1}^{F+1} c_{s_i} \propto \sum_{i=1}^{F+1} GI_i \tag{5.2}$$

where κ_b is the total computational cost of the serial fault simulation technique. The number of evaluations in the summation is $F+1$ because a fault free simulation is required before F fault simulations are performed. If fault dropping is not employed then Equation (5.2) reduces to

$$\kappa_b \propto G(F+1)I \propto G(\alpha(G))(\beta(G)) \propto O(G^3) \tag{5.3}$$

Thus, serial fault simulation without fault dropping requires that each point in the entire test volume depicted in Figure 3.2 be evaluated. Adding fault dropping decreases the amount of computation required by the serial fault simulation technique. The amount of reduction is highly dependent on the structure of the DUT and the order of evaluation of the test pattern set.

The aforementioned derivation assumes: (1) the DUT is a combinational circuit, (2) a compiled simulation approach is employed. Thus, the computational cost of serial compiled fault simulation is $O(G^3)$. Likewise, the computational cost of serial event driven fault simulation is $O(G^2)$ [91, 225, 226].

Serial fault simulation can be performed at any abstraction level. The preceding discussion assumed a gate level model. Using a higher-level model to perform fault simulation reduces the computational cost of the fault simulation. High-level fault simulation, however, requires the use of functional fault models versus low-level fault models such as the stuck at fault model. Func-
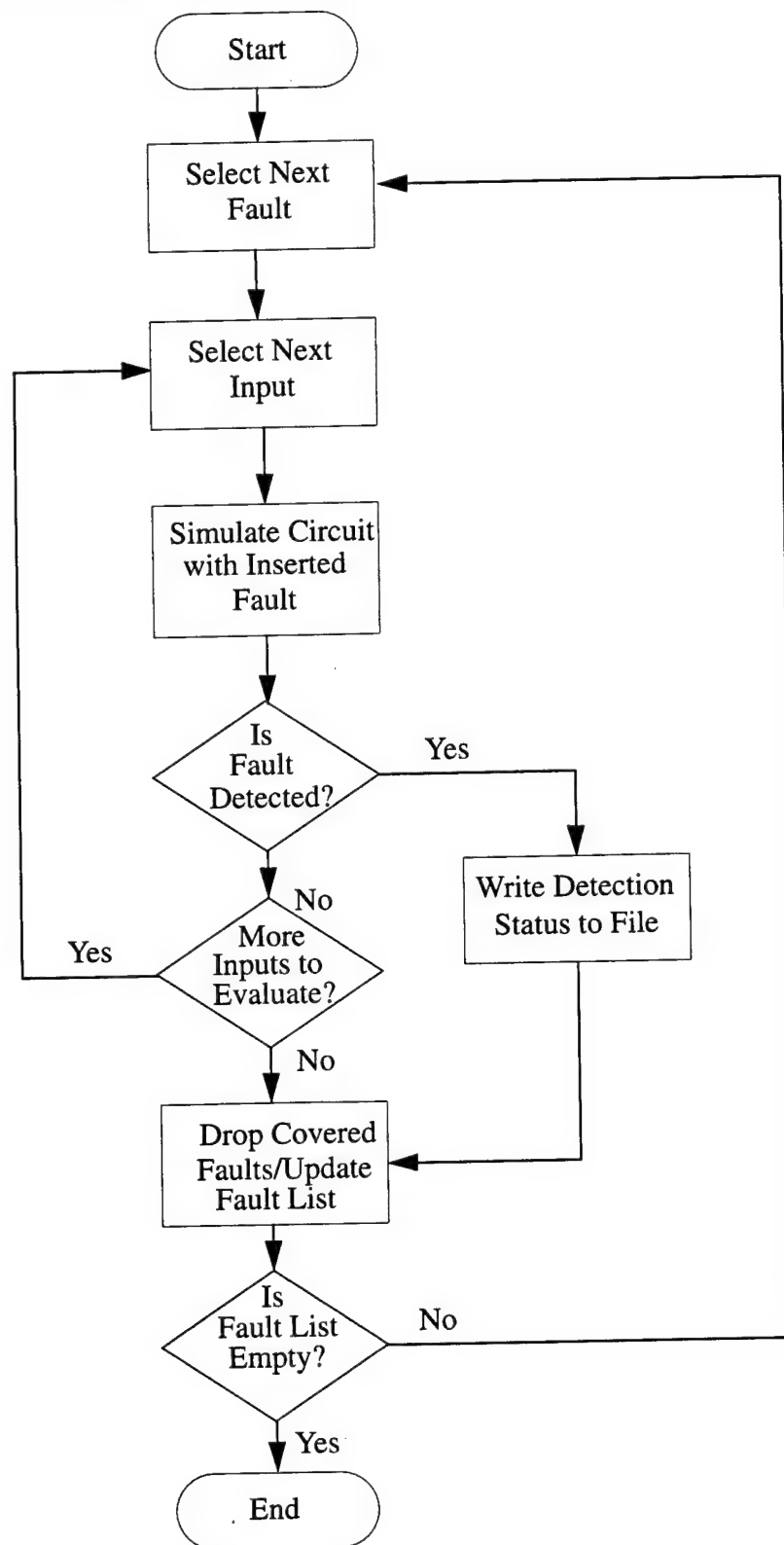
Figure 5.1.   Serial fault simulation algorithm with fault dropping.

tional fault models are used to represent fault conditions which are difficult to represent with a single low-level fault. A serial fault simulation technique which utilizes high-levels of abstraction and a functional fault model is presented in [77]. Conversely, a switch-level serial fault simulation technique which uses a stuck-on/stuck-open fault model is described in [112].

Several techniques are employed to reduce the computational complexity of fault simulation. Exploiting architectural features of the host machine that is executing a fault simulation algorithm is one approach. Parallel fault simulation is the primary technique which uses this approach and is presented in Section 5.2. Another technique is to minimize the amount of simulation required between a fault-free and a faulty DUT simulation. The foundation for this approach is that a fault in the DUT typically affects only a small percentage of the gates in the circuit. Thus, the simulation of a fault-free circuit and a faulty circuit is almost identical.

A significant savings in the number of gate evaluations required for fault simulation is achieved if a method can be developed that requires only the portion of the circuit which is affected by a fault to be reevaluated. Deductive, concurrent, and differential fault simulation techniques use this approach and are presented in Sections 5.3, 5.4 and, 5.5, respectively. Decreasing the number of gate evaluations required for a fault simulation is another speedup technique. Incorporating hierarchy into the DUT so that only a minimum number of elements are evaluated per fault simulation is one approach to reducing the fault simulation computational complexity. The exploitation of hierarchy in the DUT for efficient fault simulation is referred to as hierarchical fault simulation and is presented in Section 5.6. The other general method for reducing the amount of gate level evaluations is achieved by exploiting the inherent structure of the DUT and is referred to as Circuit Structure Based (CSB) fault simulation. There are a wide variety of CSB fault simulation techniques and an overview is provided in Section 5.7. An overview of fault simulation techniques for finite state machines which have an undefined initial state is included as Section 5.8. Additionally, a variety of fault simulators have been constructed that use a combination of the aforementioned methods. Fault simulation using a mixture of techniques is referred to as hybrid fault simulation and is described in detail in Section 5.9. An overview of existing fault simulation techniques which utilize VHDL models in some form is presented in Section 5.10.

5.2. Parallel Fault Simulation

Parallel fault simulation is designed to exploit the parallel structure of the host uniprocessor which is executing the fault simulation. One fundamental constraint of parallel fault simulation is that the DUT is simulated at the gate level and that the circuit is combinational logic. Each gate is assumed to have zero delay and all signal values are either a digital 1 or 0. Simulations which adhere to this set of assumptions are referred to as two-value zero-delay simulations. The increase in fault simulation efficiency for parallel fault simulation is achieved by noting how Boolean gates

map to the host machine. For example, each two-input, one-output gate, such as an AND, OR, and XOR, typically maps to a single host machine instruction. The instruction typically performs the desired Boolean operation using W-bit input and output operands; that is, W-bit memory locations. Only one bit of the W-bit machine word is necessary to evaluate a single logic gate. Conceptually, the execution of the instruction can be envisioned as evaluating W gates in parallel. Parallel fault simulation exploits the machine word width to evaluate W DUT in parallel.

The DUT typically requires G Boolean machine instructions to be evaluated for the circuit to be completely simulated, assuming each gate in the DUT has two inputs and one output. Only one bit of the W-bit wide word is used to simulate a single DUT. Thus, one can simulate W DUTs in parallel at the same computational cost as simulating a single DUT. Using a full host machine word in this fashion is referred to as parallel fault simulation [1, 48, 49, 52, 70, 82, 84, 89, 124, 128, 129, 142, 163, 180, 184, 187, 190, 201, 204, 211, 225, 226, 227, 229, 235].

There are two different accepted techniques for parallel fault simulation. The techniques differ in how the parallelism of the simulation is exploited. Parallel Pattern Single Fault Propagation (PPSFP) fault simulation evaluates DUTs with W different test patterns [1, 124, 187, 225, 226, 227, 229]. The second parallel technique simulates $W - 1$ faulty machines and one good machine in parallel [1, 48, 49, 52, 70, 82, 84, 89, 128, 129, 142, 156, 163, 180, 184, 190, 201, 204, 211, 235]. The good machine is used as a reference to determine if any of the $W - 1$ faults are detected. This type of parallel fault simulation is referred to as Single Pattern Multiple Fault Propagation (SPMFP). The PPSFP and SPMFP techniques are described in detail in Sections 5.2.1 and 5.2.2. Extensions to the general parallel fault simulation techniques are presented in Section 5.2.3.

5.2.1. Parallel Pattern Single Fault Simulation

Before PPSFP fault simulation is performed a fault-free simulation of W identical DUTs is performed with a unique test pattern applied to each DUT. The detected status of the injected fault is determined by comparing the fault-free parallel outputs to the PPSFP output. A discrepancy between the fault-free output and the PPSFP output indicates the injected fault is detected. A graphical example of a sample circuit which is evaluated with four parallel bit patterns is included as Figure 5.2. The example DUT has three inputs with the following parallel patterns 1101, 1011, and 1000. A single s-a-0 fault represented by an $\times$ is applied to the output of an AND gate in Figure 5.2. There are two input patterns which activate the fault and produce an error; that is, the first and last pattern. The erroneous signal values are indicated by a '*' symbol in Figure 5.2. The two errors produced by the fault are propagated to the output and are detected. In some applications W can be as large as 256 [227].

The low-level details associated with the typical PPSFP technique are described next. The first step consists of simulating the fault-free circuit with W parallel patterns and storing each W-bit
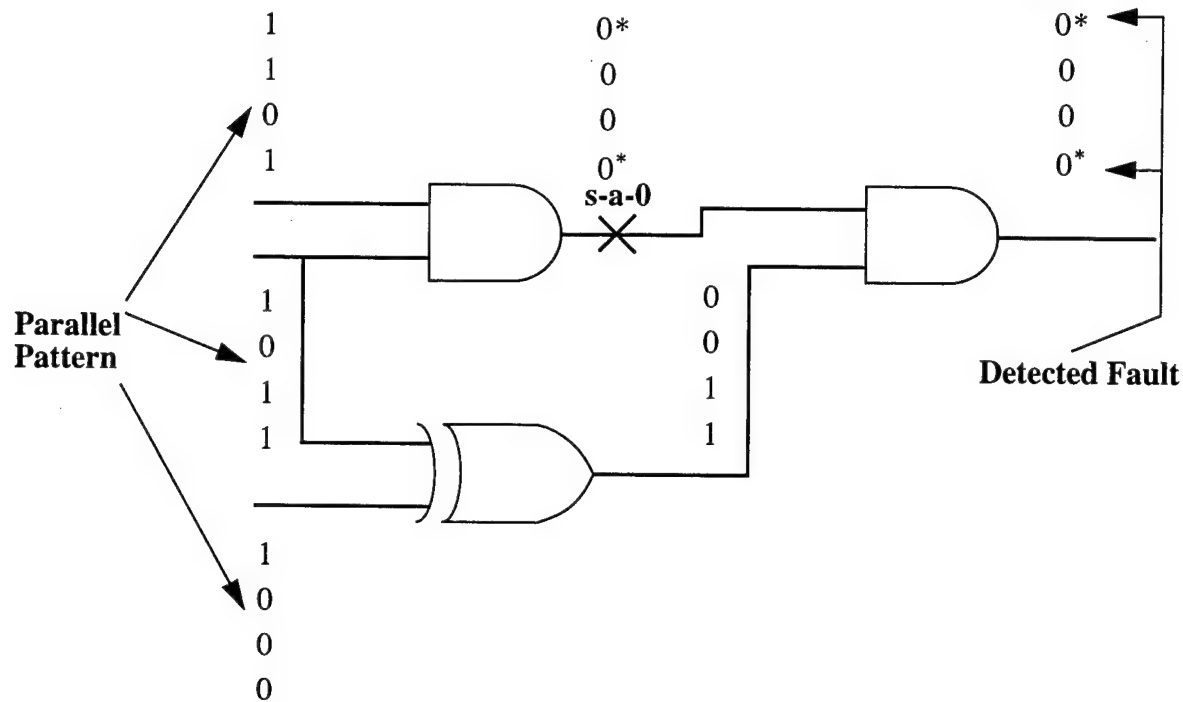
Figure 5.2.   Parallel Pattern Single Fault Propagation (PPSFP) example.

signal value. A single fault is injected at a signal location in the DUT. The erroneous value caused by the injected fault is propagated through the circuit by evaluating the gates attached to the erroneous signal. Each gate is evaluated in response to an event; that is, the propagation of the erroneous signal. If the output of a gate is erroneous then the gates which utilize the erroneous value as an input are scheduled for evaluation. The simulation of gates is complete when the evaluation list becomes empty. This type of fault simulation is referred to as event driven parallel pattern fault simulation [36, 42, 52, 86, 103, 225, 226, 227]. The evaluation list becomes empty when either the effect of the erroneous signal is masked and the fault is undetectable or the erroneous signal propagates to one or more outputs. The primary advantage of event driven parallel pattern fault simulation is that only the gates which have erroneous inputs are evaluated. Most faults affect only a small percentage of the gates in a DUT, thus event driven parallel fault simulation is more computationally efficient than compiled parallel fault simulation. In fact, event driven fault simulation has a computational cost of $O\left(G^2\right)$ [91, 225, 226]. Typically, PPSFP simulation is implemented using event driven simulation. One exception to the general trend is a technique developed by Daehn [58] where a compiled PPSFP fault simulation approach is described.

Additionally, the width of the parallel fault simulation can be dynamically increased as the number of undetected faults decreases. More machine words are added to the parallel pattern to increase the probability of detecting a single fault. The increase in machine words only adds a small amount of overhead while increasing the detection probability associated with a single fault simu-

lation. The increase in detection probability is needed when the majority of undetected faults are random pattern resistant; that is, hard to detect faults [58].

5.2.2. Single Pattern Multiple Fault Simulation

The second parallel technique simulates $W - 1$ faulty machines and one good machine in parallel [1, 48, 49, 52, 70, 82, 84, 89, 128, 129, 142, 156, 163, 180, 184, 190, 201, 204, 211, 235]. The good machine is used as a reference to determine if any of the $W - 1$ faults are detected. This type of parallel fault simulation is referred to as Single Pattern Multiple Fault Propagation (SPMFP). An example DUT with $W = 4$ is included as Figure 5.3 to further illustrate this concept. The single test pattern for the example circuit is 110. Likewise, the three faults for the example are: s-a-0 fault on each input of a two input AND gate and a s-a-1 fault on the input of an XOR gate. Each faulty DUT is assigned a unique bit position in the 4-bit machine word. The fault-free value is assigned to the lowest bit position. The two s-a-0 AND gate faults are assigned to the middle two bit positions. Likewise, the s-a-1 XOR fault is assigned to the uppermost bit position as indicated by Figure 5.3. The erroneous signals produced by the faulty machines are indicated by the '*' symbol in Figure 5.3. Thus, with this example three faulty DUTs are simulated simultaneously with a fault-free DUT. Each faulty DUT contains one fault from the fault list. Also, for the example depicted in Figure 5.3 all three faulty DUTs are detected because each faulty DUT produces an erroneous output.
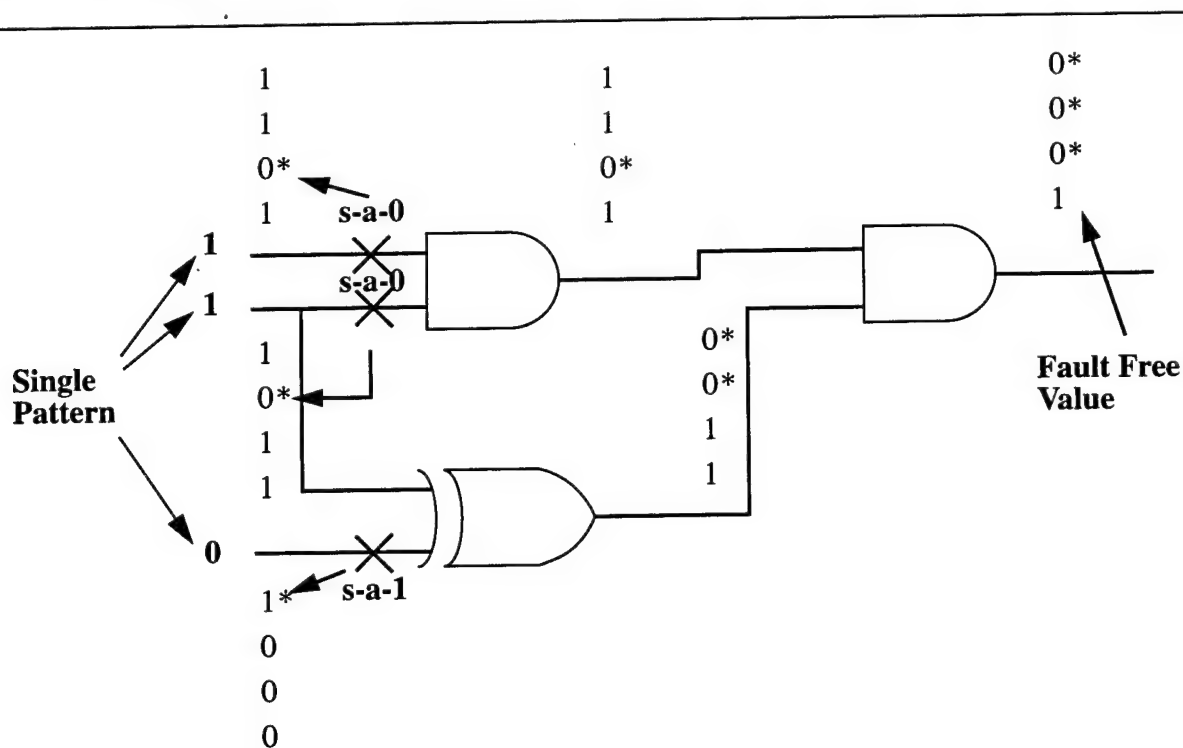
Figure 5.3.   Single Pattern Multiple Fault Propagation (SPMFP) example.

Typically SPMFP simulation uses a compiled simulation approach which consists of evaluating all G gates for each SPMFP fault simulation and is referred to as compiled parallel fault simulation [24, 57, 107, 118, 225, 226, 235]. The generally accepted computational cost of compiled SPMFP fault simulation is on the order of $O(G^3)$ given that every gate is evaluated when a new input pattern is applied during fault simulation [80, 178, 225, 226]. Thus, the cost associated with compiled SPMFP fault simulation is essentially equivalent to compiled serial fault simulation. In fact, compiled parallel fault simulation is at most a factor of W more efficient than serial fault simulation. Reducing the computational cost of an $O(G^3)$ algorithm by a constant results in an $O(G^3)$ computational cost. Thus, compiled SPMFP fault simulation provides an incremental improvement in fault simulation efficiency.

Typically, event driven SPMFP is not employed for fault simulation. The complexity involved with simulated asynchronous circuits with event driven SPMFP can be quite large. The cause for the large overhead is directly related to parallel faults being evaluated on an asynchronous circuit. Specifically, each fault condition can cause different gates in different parts of the DUT to propagate vastly different errors. Each error condition can in theory cause the number of state transitions for each faulty DUT to reach a steady-state condition to be different. While each bit position in the computer word stores the faulty behavior for each individual faults the asynchronous feed back present in asynchronous finite state machines introduces the additional overhead. For example, the fault associated with the ith bit position can cause an error to be produced which propagates to a feed back path. Considered the case where the fault free DUT requires three state transitions for the asynchronous finite state machine to reach steady-state. For the sake of discussion, assume that the fault which causes the feedback path to be erroneous causes five state transitions for the asynchronous finite state machine to reach steady state value. The overhead occurs when the fourth and fifth state transition of the asynchronous machine is evaluated for the faulty machine. The extra two evaluations can cause the state of the other faulty DUTs to be set to an incorrect value. Specifically, one fault can cause the feedback loop for all faulty machines to be evaluated five times versus three times. Extra book keeping is required to assure that only the number of cycles required to reach steady-state for each of the parallel faulty DUT is performed. The book keeping required for the general case of asynchronous DUTs is quite large. Due to the large overhead the event driven SPMFP fault simulation technique is typically not used to evaluate asynchronous finite state machines.

5.2.3. Extensions to Parallel Fault Simulation

The two-value zero-delay combinational circuit parallel fault simulation algorithm which uses the stuck-at fault model has been extended in several ways. One major extension to parallel fault simulation is the ability to simulate sequential circuits [86, 123]. The basic concept is to have the

state of the sequential circuit always start at an unknown state except for the first input vector. A graphical depiction of the technique is included as Figure 5.4 to facilitate discussion. The first test pattern is given as $I[0]$, while the initial state is given as $S[0]$ in Figure 5.4. The initial state for $S[0]$ is known while all other state values are not known. Thus, $S[0]$ is initialized to some value before the DUT is evaluated. The additional parallel test patterns are stored in vectors $I[1]$ to $I[W-1]$. The $I[1]$ to $I[W-1]$ input vectors represent the 1 to $W-1$ input patterns which are applied sequentially to the physical DUT. The state vector associated with the sequential circuit after $I[0]$ is evaluated is $S[1]$. The value of $S[1]$ is known after the first evaluation of the DUT. However, the $S[2]$ to $S[W-1]$ state vectors are not completely known after the first evaluation because the initial value of $S[1]$ to $S[W-1]$ are unknown. Three-value logic is employed to initialize $S[1]$ through $S[W-1]$ to an all X value. Additionally, three-value logic is used to simulate the DUT for all W input vectors in parallel. Using three value logic; that is, 1, 0, and X for unknown value, one is able to potentially determine a portion of the next state of the machine for each input vector; that is, the $S[2]$ to $S[W-1]$ state vectors are partially specified after the first evaluation. Conceptually, this parallel sequential technique is attempting to evaluate the next state and output of a sequential machine for W test patterns simultaneously. The state information from the first input vector ($S[1]$) is used on a second pass to fully determine the state of the sequential circuit for the second input vector ($S[2]$) and to more fully define the remaining partial state vectors associated with the other input vectors ($S[3]$ to $S[W-1]$). Additional passes are made until either all states in the simulation are resolved or the inserted fault is detected [86, 123].

Lee presents a PPSFP parallel fault simulation algorithm which utilizes three value logic and preprocessing of the DUT to exploit the properties inherent in the DUT to speed fault simulation [127]. Utilization of the inherent structure of the DUT to speed simulation is referred to as circuit structure based techniques and an overview is provided in Section 5.7.

The use of fault models other than the stuck-at fault model is another extension to parallel fault simulation. The use of a delay fault model with the parallel fault simulation technique is described in [42, 118]. Hwang incorporates switch-level and gate-level fault models into a single parallel fault simulation algorithm in [103]. Conversely, Mahlstedt presents a parallel fault simulation technique which incorporates the following fault models: (1) stuck-at, (2) function conversion, (3) bridging, and (4) transition [138]. The fault simulation methodology allows for single faults of any of the four fault models and multiple faults using any combination of the four models. Waicukauski presents a PPSFP technique which utilizes a transition (delay) fault model [229]. Another extension of the parallel fault simulation algorithm allows for the DUT to contain tristate devices and bidirectional circuits [219].

One of the primary advantages of parallel fault simulation is the low-level of overhead associated with the technique. Most other fault simulation techniques are far more complex than parallel

Inputs

$I[W-1]$

$I[W-2]$

$I[W-3]$

$I[2]$

$I[1]$

$I[0]$

Combinational
Logic

$S[W-1]$

$S[W-2]$

$S[W-3]$

$S[2]$

$S[1]$

$S[0]$

State

Outputs

$O[W-1]$

$O[W-2]$

$O[W-3]$

$O[2]$

$O[1]$

$O[0]$
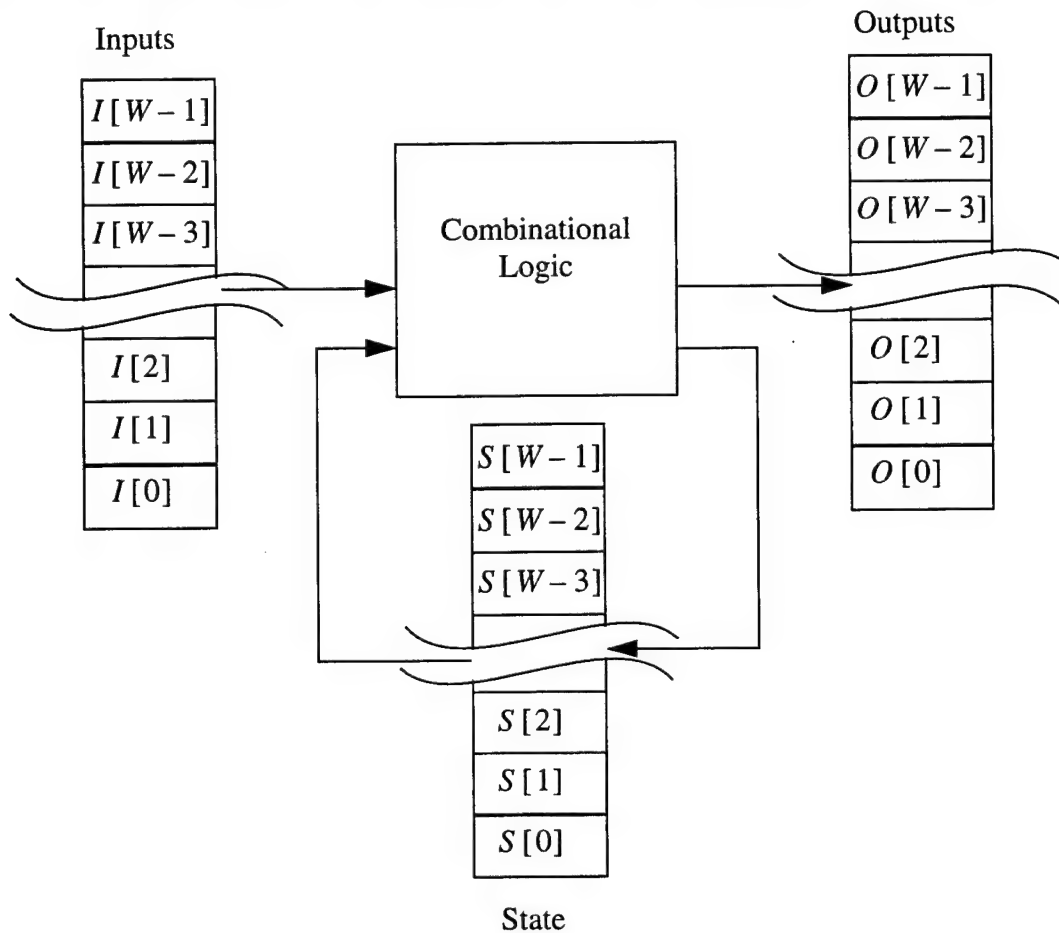
Figure 5.4. Example sequential circuit evaluated via PPSFP fault simulation.

fault simulation. There is a large amount of debate on the optimal fault simulation technique. Some experts argue that event driven PPSFP fault simulation on a large fast host machine is the best fault simulation algorithm [3]. Additionally, since parallel fault simulation occurs at the gate level the use of special purpose hardware logic accelerators is a viable option to improve the performance of parallel fault simulation. Hardware accelerators are typically used to speed simulation times during the design of the DUT. An overview of hardware accelerators is provided in Section 7.

5.3. Deductive Fault Simulation

The deductive fault simulation technique simulates the behavior of the good circuit for a given input vector and deduces the behavior of faulty circuits. The deductive process implies a theoretical capability that may or may not be realized due to memory limitations of the host processor. Given that the host processor has enough resources then two items are produced at the end of a deductive fault simulation; that is, the correct outputs and the list of faults which are detectable. The deductive technique requires that a fault list be associated with each signal line. Each fault list contains

the set of equivalent faults which causes the signal associated with the fault list to be erroneous. A conceptual diagram depicting a two-input AND gate and the associated fault lists is included as Figure 5.5. The input fault lists for a given gate are evaluated using set theory operations to produce a gate output fault list. Unlike parallel fault simulation the deductive method is capable of handling combinational, synchronous, and asynchronous circuits. Similarly, the deductive technique relies on a two-value zero-delay gate level simulation that uses the stuck-at fault model [1, 19, 48, 82, 84, 117, 128, 129, 142, 163, 174, 180, 202, 208].

The deductive fault list propagation technique uses the concept of a controlling input. An input is defined to be controlling if changing the value of the input causes the output value of the gate to change. For example, assume that the AND gate depicted in Figure 5.5 has $A = 0$, $B = 1$, and $C = 0$. For this example A is a controlling input while B is a noncontrolling input. A hypothetical fault list associated with A is given as

$$L_A = \{a_1, p_1, q_0, r_1\} \tag{5.4}$$

where L_A is the set of faults associated with A, a_1 is a s-a-1 fault on A, p_1, q_0, and r_1 are stuck-at faults on other signals in the DUT which cause A to be erroneous. Likewise, a hypothetical fault list for B is

$$L_B = \{b_0, q_0, r_0\} \tag{5.5}$$

The output fault list associated with C is calculated by evaluating
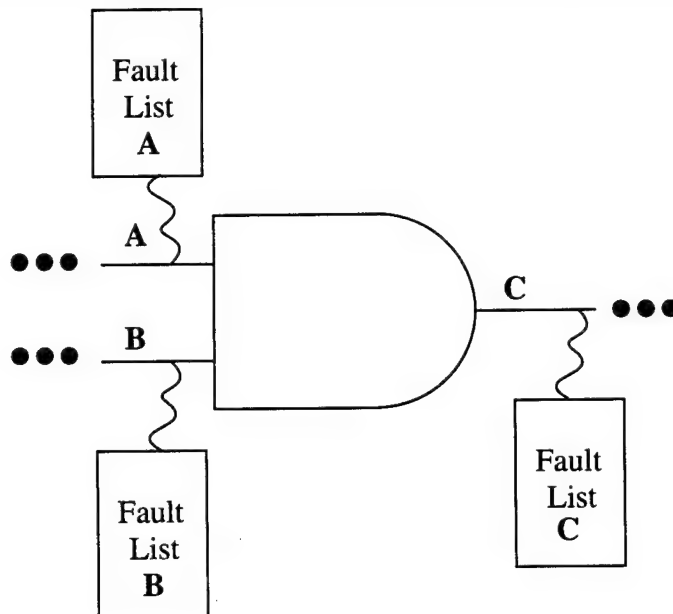
Figure 5.5. Deductive fault simulation diagram.

$$L_C = \{c_1\} \cup (L_A - L_B) = \{c_1, a_1, p_1, r_1\} \tag{5.6}$$

A graphical depiction of the set subtraction operation is included as Figure 5.6 to illustrate the evaluation of Equation (5.6). The output fault list equation is derived by noting that faults associated with the controlling input which are not contained in the noncontrolling input fault list propagate to the output of the gate. The stuck-at fault associated with the output of the gate that produces an error is also added to the output fault list.

The general form of the fault list propagation equation is derived by extending the concepts presented in the previous example. The derivation begins by assuming that a gate has n inputs which can be divided into two mutually exclusive sets, controlling and noncontrolling inputs. The controlling input set is referred to as I_c while the noncontrolling input set is called I_{nc}.
The controlling input sets comprise all single controlling inputs and multiple controlling input sets associated with a given component. For example, a four input OR gate with a 0110 input pattern has one set of multiple controlling inputs; that is, when the 11 inputs are changed to a 00 value. The input pattern to the component defines which controlling input set is applicable for a given evaluation. It is possible for a propagated fault to appear on any one or multiple inputs to a component. For a fault to propagate through a component it must appear only on the inputs which are in the controlling input set. If the fault appears on any member of the noncontrolling input set then the fault cannot propagate. The general form of the output fault list calculation is given by
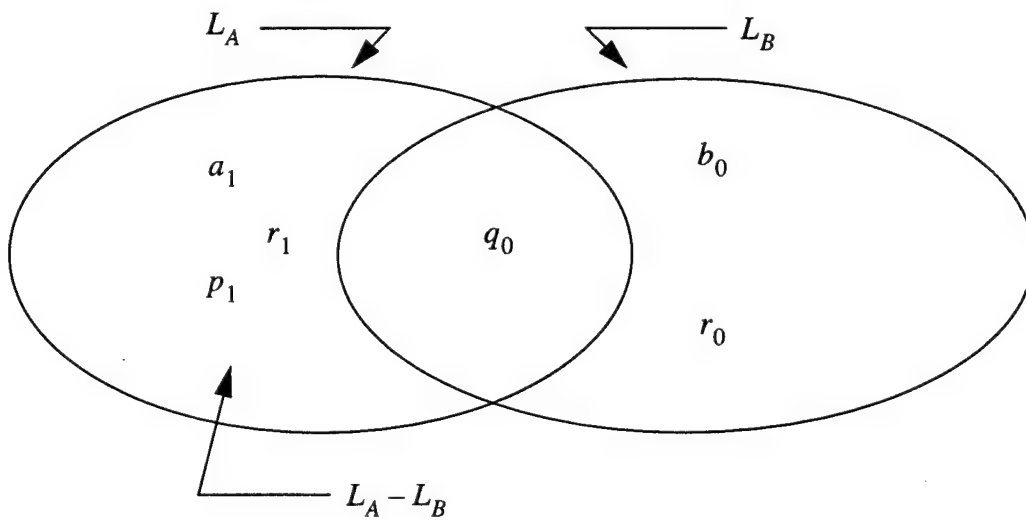
Figure 5.6. Graphical depiction of set subtraction operation specified by Equation (5.6).

$$L_O = \{O_{s\text{-}a\text{-}x}\} \cup \begin{cases} \displaystyle\bigcup_{i \in I_c} L_i \forall (I_{nc} = \varnothing) \\ \displaystyle\left(\bigcap_{i \in I_c} L_i - \bigcup_{i \in I_{nc}} L_i \right) \forall (I_{nc} \neq \varnothing) \end{cases}$$ (5.7)

where O is the output line of the gate, $\varnothing$ is the null set, and $O_{s\text{-}a\text{-}x}$ is the output stuck-at fault which produces an error [1, 142, 174].

The fault simulation begins by placing one stuck-at fault in each fault list associated with the input to the DUT. Each gate in the circuit is evaluated using an event driven approach. Essentially, a gate is scheduled to be evaluated when any inputs to the gate are set to some value. The gate evaluation process consists of three steps: (1) calculate the output, (2) determine the set of controlling and noncontrolling inputs, and (3) create the output fault list. A fault simulation is complete when there are no gates scheduled to be evaluated for the DUT.

The deductive fault simulation has two types of activity, true value changes and fault list changes. A true value change results in the recalculation of fault lists. However, it is possible for fault lists to be updated without a value change. A gate is scheduled for evaluation when either the true value or the fault list associated with the gate's inputs are updated. Under certain conditions, it is possible for a deductive simulation to calculate an inordinate number of fault list updates. This type of behavior typically occurs in asynchronous circuit simulation and is analogous to an asynchronous circuit failing to reach a steady-state value. When this fault list or state settling problem occurs, the deductive fault simulation takes an inordinate amount of simulation time [1, 174].

Deductive fault simulation can also utilize three-value logic to facilitate the simulation of circuits which contain state and are not initialized to a predefined state. Under certain fault conditions, the state initialization circuitry for the DUT may fail to initialize the device to a predefined state. Three-value deductive simulation is approximate in nature since it may fail to report some detectable faults as detected [1, 174]. The details associated with three-value deductive simulation are not presented here due to space limitations.

A recent development by Smith attempts to increase the efficiency of deductive simulation by eliminating the evaluation of gates which do not affect the output of the DUT [202]. The basic idea is to start the simulation at the output of the DUT. The gate which drives the output is evaluated by using a recursive procedure. The evaluation procedure determines that the inputs to the current gate are unknown, locates the gates which set the input value, and recursively calls the gate evaluation procedure to evaluate the gates which set the input value. The recursion ends when the inputs to the DUT are reached. This type of simulation is referred to as demand driven simulation [202]. Only the gates which affect the output are evaluated with the backward evaluation. Once all of the inputs are known for a particular gate then the deductive simulation algorithm is performed. Specifically, each gate is evaluated using the following three steps: (1) calculate the output, (2) deter-

mine the set of controlling and noncontrolling inputs, and (3) create the output fault list. The demand driven deductive simulation is used to evaluate synchronous circuits [202]. Additionally, the gates in the simulation are assumed to have a unit delay propagation.

Another extension to the deductive fault simulation technique is the ability to simulate multiple faults simultaneously. For multiple fault simulation via the deductive technique to be feasible the multiple fault conditions must be represented in a minimal fashion. Utilizing codeword concepts to store multiple fault conditions is one approach. Each fault scenario associated with a DUT is represented by a unique codeword. The codeword generation process begins by listing all single and multiple faults associated with a DUT. The list of faults is given as

$$F = \{f_0, f_1, ..., f_{n-2}, f_{n-1}\} \tag{5.8}$$

where n is the number of total faults. Each fault is assigned a distinct codeword which represents a Boolean function. A v-bit codeword given as

$$c_k = c_{k_0} c_{k_1} ... c_{k_{v-1}} \tag{5.9}$$

is used to represent a fault, where $v > \log(n)$. A characteristic function Φ_F is used to generate the Boolean functions which represent the codewords and is given as

$$\Phi_{\{f_k\}} (x_0, x_1, ..., x_{v-1}) = \xi_0 \cdot \xi_1 \cdot ... \xi_{v-1}$$
$$\text{where} \begin{cases} \xi_i = \bar{x}_i \text{ if } c_{k_i} = 0 \\ \xi_i = x_i \text{ if } c_{k_i} = 1 \end{cases} \tag{5.10}$$

To further illustrate the codeword generation process consider the case where $F = f_0, f_1, ..., f_7$. The codeword for each member of the fault set is represented by Boolean functions and is given as

$$\Phi_{\{f_0\}} = \bar{x}_2 \bar{x}_1 \bar{x}_0, \Phi_{\{f_1\}} = \bar{x}_2 \bar{x}_1 x_0, ..., \Phi_{\{f_7\}} = x_2 x_1 x_0 \tag{5.11}$$

The set operations associated with single fault deductive simulation are modified to propagate the codeword fault representation appropriately. Specifically, the deductive technique requires numerous set union and intersection operations to be performed. With the Boolean function codeword representation the set union and intersection operations are performed using Boolean operators. The union operation is given as

$$\Phi_{A \cup B} = \Phi_A + \Phi_B \tag{5.12}$$

Likewise, the intersection operation is defined as

$$\Phi_{A \cap B} = \Phi_A \cdot \Phi_B \tag{5.13}$$

The Boolean function representation of the faults allows for the multiple fault scenario to be compactly represented and also allows the set intersection and union operations required by deductive fault simulation to be performed efficiently [208].

The computational cost of deductive fault simulation is $O(G^2)$ [91, 178]. Thus, deductive fault simulation has a similar computational complexity as event driven PPSFP fault simulation. The deductive technique requires significantly more memory to execute than PPSFP fault simulation. The increase in memory is due to the fault lists associated with the signal lines. Additionally, since the fault lists grow in size dynamically it is very difficult to determine the amount of memory required for deductive fault simulation for a given DUT and input pattern. The updating of the fault lists also introduces computational overhead which is not present with parallel fault simulation. Deductive fault simulation, however, requires only one simulation per input vector to locate all detectable faults. Thus, the increase in overhead associated with calculating and updating the fault lists is offset by the single evaluation of an input pattern for the deductive technique.

The primary limitation of deductive fault simulation is the large amount of overhead associated with performing the set operations which are used to propagate the fault list [1, 174]. The cost associated with the set operations becomes higher as the number of inputs to a component increase. Specifically, components which have a large number of inputs require a large number of set operations to be performed to propagate the component fault list. For this reason, the deductive technique is rarely used at abstraction levels higher than the gate-level. As designs become more complex it is desirable to have the ability to perform fault simulation on a model which has certain elements described using a high-level functional model while other portions of the DUT are represented at the gate level. The concurrent fault simulation method, described in the next section, is a more efficient mixed level fault simulation technique as compared to the deductive method.

5.4. Concurrent Fault Simulation

Concurrent fault simulation exploits the fact that the signal activity for a good DUT and a faulty DUT is virtually identical. Like deductive fault simulation, concurrent fault simulation is theoretically capable of locating all detectable faults with a single simulation. The capability of locating all detectable faults is dependent on the resources of the host computer executing the concurrent fault simulation. The concurrent technique can require a large amount of memory for a given simulation. It is difficult to determine the amount of memory required *a priori* for a given DUT and test pattern. The concurrent technique supports the simulation of combinational, asynchronous, and synchronous circuits. The stuck-at fault model with two-value logic is used with the concurrent fault simulation technique. The primary advantage of concurrent fault simulation versus deductive is the support of models which incorporate mixed levels of abstraction [1, 2, 31, 32, 39, 40, 46, 55, 82, 95, 125, 128, 129, 132, 174, 180, 186, 189, 195, 196, 200, 214, 216, 217].

The concurrent technique stores the active fault list in the form of faulty components and each component can be at a different level of abstraction. The concurrent technique is defined by how the fault list associated with each component is calculated and propagated. A simple three gate

example is included as Figure 5.7 to illustrate the concurrent fault simulation algorithm. The inputs to the example circuit are $A = 0, B = 1, C = 1, D = 1$. The fault list for the AND gate with the A and B inputs is evaluated first. The concurrent fault simulation algorithm constructs the fault list associated with the component. For the upper AND gate in Figure 5.7 the fault list consists of A s-a-1 (A_1) and B s-a-0 (B_0). The A_1 fault causes the AND gate to produce an erroneous 1 output and is propagated to all components which use the output signal as an input. The second step of the concurrent fault simulation algorithm is the evaluation of the AND gate which has the C and D inputs. The fault list for this component is then constructed. The fault list consists of C_0 and D_0 both of which cause the output signal of the AND gate to be erroneous. For this reason, both faults are propagated with the output value associated with the lower AND gate in Figure 5.7. The outputs of the upper and lower AND gates are given as E and F in Figure 5.7. The third com-
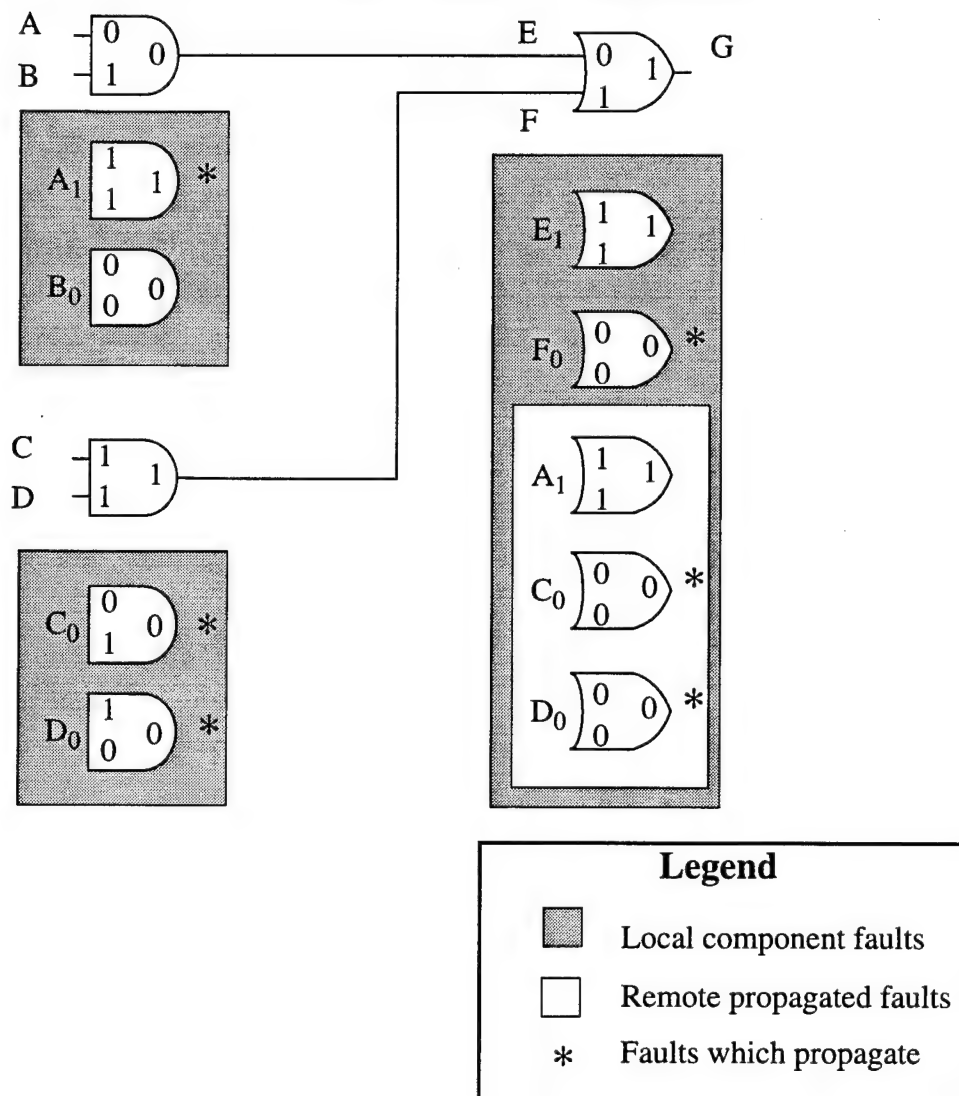
Figure 5.7. **Concurrent fault simulation algorithm circuit example.**

ponent evaluated in this example is the OR gate on the right of Figure 5.7. The fault list construction for the OR gate begins by adding the input faults associated with the OR gate to the fault list. For this example the input faults are E_1 and F_0 with the F_0 fault causing G to be erroneous. The propagated faults are then added to the OR gate fault list. For this example, there are three propagated faults; that is, A_1, C_0, and D_0. The C_0 and D_0 faults cause the output signal G to be erroneous. The concurrent fault simulation algorithm ends by indicating that F_0, C_0, and D_0 are detected for the current input vector depicted in Figure 5.7.

A generic example component with a hypothetical fault list is included as Figure 5.8 to further assist in illustrating the concurrent fault simulation technique. The example component is a two input AND gate which has a 01 input vector applied. The fault-free output for this gate is calculated first and results in a logical 0 being produced. The fault list construction for the example gate in Figure 5.8 begins by adding the faulty gates which are associated with the input signals. For this particular example there are two faults which propagate with the inputs and are referenced as α and β. Each fault in the DUT is given a unique label. The input component faults which could potentially cause the output of the component to be in error are then added to the fault list. For this particular example, the component is an AND gate which has a zero output. The set of input faults which could cause the output to change are s-a-1 faults. The two s-a-1 input faults are added to the
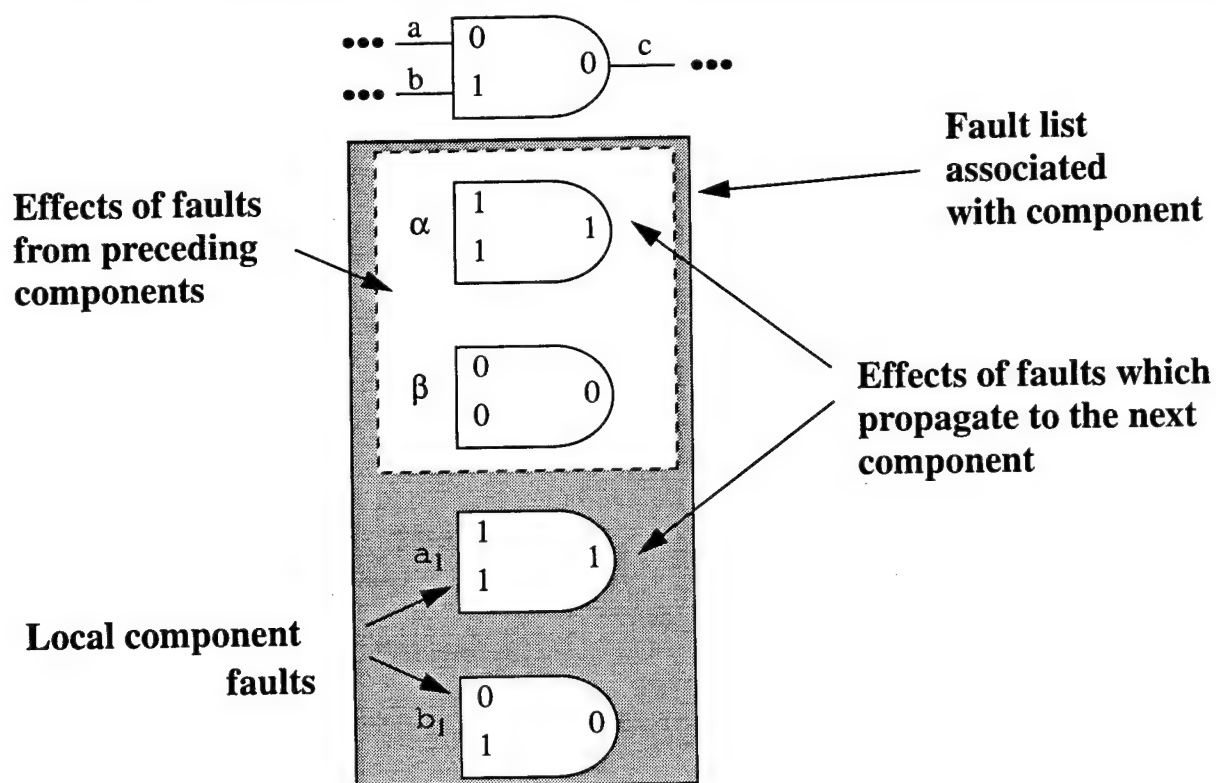
Figure 5.8. Concurrent fault simulation example diagram.

example fault list in Figure 5.8 as a_1 and b_1. The next step associated with the fault list construction is the calculation of the output of the component for each fault in the fault list. For the example fault list α and a_1 faults produce an erroneous output, while the β and b_1 faults produce the correct output. Only the faults which produce a component output error are propagated with the correct component signal value.

Comparing the concurrent fault list generation process with the deductive fault list generation method one notices that the concurrent fault list contains more faults than the deductive technique. For the example depicted in Figure 5.8 fault b_1 would not be included with the deductive method. The concurrent technique utilizes fault lists which are a superset of the deductive fault lists [174].

The concurrent technique can be conceptually viewed as simulating all possible faulty DUTs that could produce an error at each component. The addition of a new faulty DUT to the set of faulty DUTs occurs when the fault under consideration causes the signal value of the good DUT to diverge from the correct value. For example, fault a_1 in Figure 5.8 causes divergence. Each faulty DUT is uniquely identified by the unique fault id associated with the fault list of the DUT. A faulty DUT is dropped from the list of faulty DUTs to evaluate whenever the faulty DUT's signal value matches the correct signal value. When this condition occurs the faulty DUT is said to converge to the good DUT. The divergence and convergence of faults causes the fault list associated with each component to have a dynamic size. Thus, concurrent fault simulation requires dynamic memory allocation and deallocation. Additionally, the computational cost associated with concurrent simulation is $O(G^2)$ [91, 178]. Like deductive fault simulation, the concurrent technique has a significant overhead associated with generating and propagating the fault list for each component. However, the concurrent technique locates all detectable faults on a single simulation which more than compensates for the fault list overhead.

The concurrent method has also been extended to use fault models other than the stuck-at fault model. Specifically, Shih presents a transistor-level concurrent simulator [195, 196]. This simulator utilizes the stuck-on/off and the stuck-open fault models.

5.5. Differential Fault Simulation

Differential fault simulation is a technique to evaluate sequential circuits by extending the serial fault simulation technique. The differential technique uses two-value zero-delay concepts for the simulation of the DUT. A structural diagram of the differential method is included as Figure 5.9 to assist in describing the technique. The basic idea is to simulate the DUT with the first input vector $I[0]$, store the next state of the DUT $S_{ff}[1]$, and store the fault-free outputs $O[0]$. A fault (f_0) is then inserted into the DUT and the effect of the fault is evaluated using an event driven simulation approach. Thus, only the gates which are affected by the fault are re-simulated. If the inserted fault causes an output change then the fault is detected and the fault is dropped from the
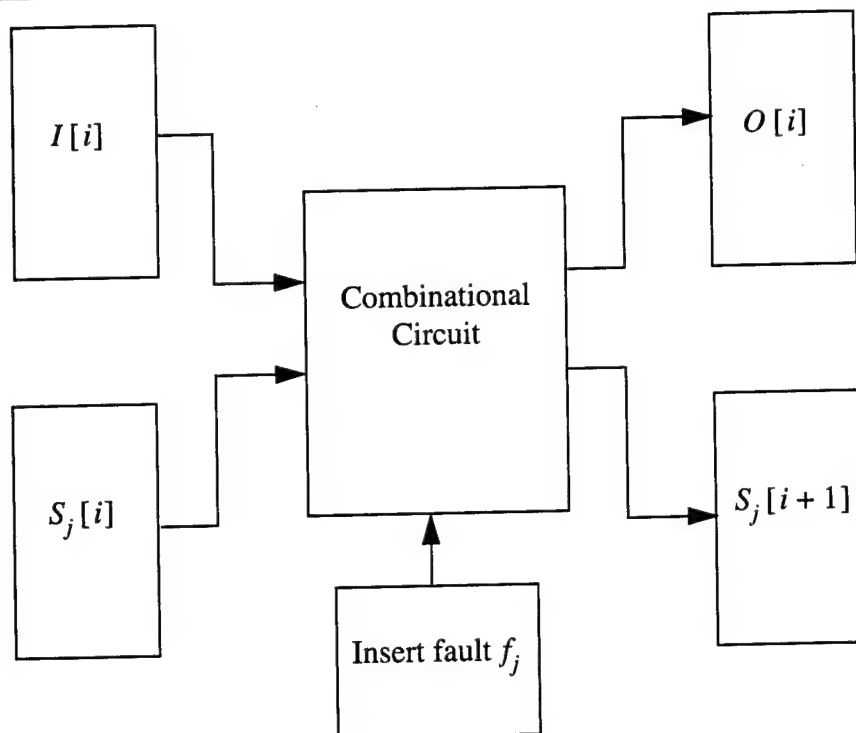
Figure 5.9. Differential fault simulation method depicting all structural components.

fault list. If the fault is not detected then the next state vector of the faulty DUT is saved as $S_0[1]$. The current fault is removed and the next fault is inserted in the DUT. Using traditional event driven fault simulation would require that all erroneous signals be removed from the DUT before the next fault is inserted into the DUT. Removing the erroneous signals is referred to as signal restoration. The differential fault simulation technique does not perform the restoration step. The basic philosophy is to order the fault list such that adjacent faults are likely to cause nearly identical erroneous behavior. Thus, eliminating the restoration step eliminates unneeded simulation.

The evaluation of faults continues until all the faults in the fault list are evaluated. The next input vector is applied, the stored fault-free next state value calculated with the previous input vector is restored, and the next state and correct output is calculated and stored. The fault list for this evaluation contains only the faults that were not detected by the previous input vector. For the general evaluation of the jth fault with the ith input the differential fault simulation method reads the $S_j[i]$ state value and the $I[i]$ test pattern, inserts the fault f_j, performs the simulation, and stores the next state vector $S_j[i+1]$ if the fault does not cause an output error. Thus, the evaluation of the fault list for the ith test pattern entails restoring the next state vector $S_j[i]$ from the previous input for the jth fault [51].

The differential fault simulation technique requires substantially less memory than concurrent fault simulation. The only additional information required for storage with the differential fault

simulation technique is the next state vectors for the good DUT and all faults in the fault list. Conversely, concurrent fault simulation must store the fault list for every component in the DUT.

The order in which the fault list is evaluated affects the performance of the differential fault simulation technique. If the fault list is ordered such that the faults which cause nearly equivalent erroneous events to occur then the amount of events processed by the event driven simulation is minimized. Cheng explored various orderings to determine that a depth first fault list which starts with the outputs and proceeds to the inputs provides the best performance [51]. Likewise the computational complexity of differential fault simulation differs from event driven parallel fault simulation by a factor of W. Thus, differential fault simulation is of $O(G^2)$ complexity.

The differential technique was extended by Vandris to fault simulate switch-level DUTs [220, 221]. A stuck-on/off fault model is employed for the switch-level fault simulator. Vandris demonstrates that the differential method is faster than the concurrent method for switch-level DUTs.

5.6. Hierarchical Fault Simulation

One method for improving fault simulation efficiency is achieved by using the partitioning and hierarchy which is inherent in the design of a complex device. Typically, the first step in the design of a complex device is partitioning the problem into smaller more manageable subdevices. The interface to each subdevice is defined so that each device can be designed independently. In theory, each subdevice can be designed in parallel by an independent design team. During the design process each subdevice is represented by a certain level of design abstraction. Commonly accepted terms for levels of design abstraction are: (1) functional level, (2) register transfer level, (3) gate level, and (4) switch (transistor) level. The previous list of abstraction levels starts at the highest level of abstraction (functional models) and ends with the lowest level of abstraction (switch level models). In general, the higher the level of abstraction the smaller the amount of computational cost to simulate the model. However, increased level of abstraction decreases the amount of information provided by the simulation.

One way for exploiting the inherent design structure is to simulate all subdevices, except for the one subdevice which contains the fault, at the highest level of design abstraction. Exploiting the hierarchy of the design in this fashion is referred to as hierarchical fault simulation [2, 47, 61, 72, 73, 74, 87, 103, 137, 144, 147, 148, 159, 178, 179, 180, 185]. Conceptually, using the design hierarchy in this fashion is equivalent to reducing the height of the test volume depicted in Figure 3.2 by some amount. Thus, the computational cost of each fault simulation is theoretically reduced with hierarchical fault simulation. To illustrate the reduction in computation cost consider the case where the DUT consists of 10 subdevices. Assume the computational cost of simulating each of the individual high-level subdevice models is equivalent to the computational cost associ-

ated with 50 logic gates. Conversely, assume that each subdevice gate level model contains 1,000 gates. Performing a single simulation of the DUT at the gate level has a computational cost of 10 subdevices times the simulation of 1,000 logic gates or 10,000. However, if only one subdevice is simulated at the gate level while all other subdevices are simulated at the high-level then the computation cost of one simulation is $1,000 + 9(50) = 1,450$. Thus, the simulation which uses hierarchy requires approximately 6.9 times less computational resources than the low-level simulation approach. For this simple example, the height of the test volume depicted in Figure 3.2 is reduced by a factor of 6.9.

The previous example can be generalized for the case where a DUT contains n components. Suppose each component has a gate-level model and a functional model. The computational cost measured as the number of gate evaluations for the ith component for the gate and functional model is represented as g_i and h_i, respectively. The total time required to simulate the gate-level model of the DUT is

$$T_g = \sum_{i=1}^{n} g_i T \tag{5.14}$$

where T is the time required to evaluate a gate and T_g is the total time required to simulate the gate-level DUT model. The time required to evaluate the hierarchical fault simulation model where one component is represented by a gate-level model and all other components are represented by functional models is given as

$$T_h = g_j T + \sum_{i \in \{1, 2, ..., n\} \, \forall \, (i \neq j)}^{n} h_i T \tag{5.15}$$

where T_h is the time required for hierarchical fault simulation of the jth component.

The speed up observed by exploiting hierarchy in this fashion is calculated by dividing the time required for gate level evaluation by the time required for hierarchical evaluation. Performing this division provides

$$S_u = \frac{T_g}{T_h} = \frac{\displaystyle\sum_{i=1}^{n} g_i}{g_j + \displaystyle\sum_{i \in \{1, 2, ..., n\} \, \forall \, (i \neq j)}^{n} h_i} \tag{5.16}$$

where S_u is the speed up factor associated with evaluating the jth component. The average speed up factor is desired and is calculated by using the average computational cost for evaluating the gate-level and functional-level components. Using the average computational cost per component in Equation (5.16) provides

$$S_u = \frac{ng}{g + (n-1)h} \tag{5.17}$$

where g is the average computational cost for a gate-level component model and h is the average computational cost for a functional-level component model. For large n the speed up function can be approximated as

$$S_u = \frac{g}{h} \tag{5.18}$$

Thus, for large n, the speed up with hierarchical fault simulation reaches the theoretical maximum speed up possible for exploiting hierarchy to increase the efficiency of fault simulation

An example circuit which utilizes design hierarchy is included as Figure 5.10 to illustrate the nature of hierarchical fault simulation. The example contains subdevices which are represented at the gate level and one functional level subdevice model. A s-a-0 fault is applied to one of the inputs of the functional level subdevice as shown in Figure 5.10. The mixed level model is then simulated to determine if the inserted fault produces an error on the output of the DUT; that is, the output of the right most AND gate.

There are a variety of techniques which can be used to insert faults in a hierarchical model and perform a simulation. Unfortunately, both parallel and deductive simulation cannot be extended for use with hierarchical models. Very few high-level models map to a specific host processor instruction which is the fundamental construct utilized by the parallel fault simulation technique. Also, the set theory fault propagation technique used for deductive becomes cumbersome for components which have a large input space. The most common approach to performing fault simulation on a hierarchical model is via the concurrent fault simulation technique [2, 47, 61, 72, 73, 74, 87, 103, 137, 147, 148, 159, 178, 179]. The concurrent fault simulation technique can use any level of abstraction to propagate the fault list for a given device. Concurrent fault simulation performed on
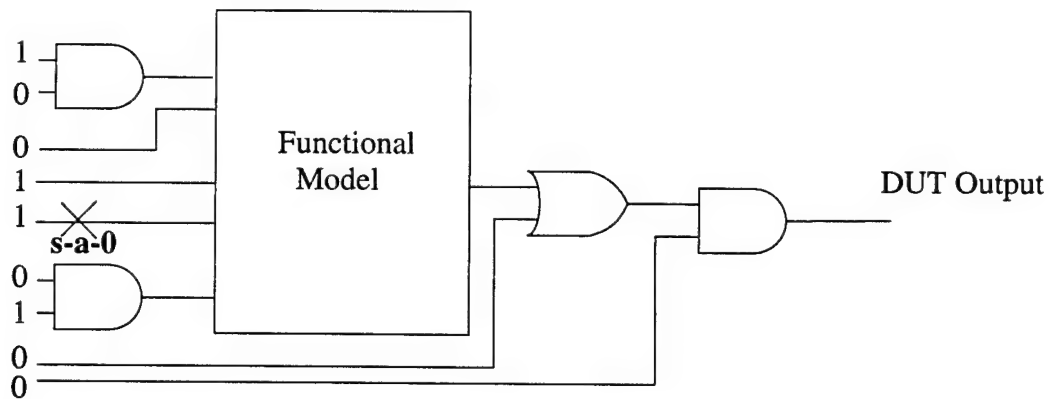
Figure 5.10. Typical hierarchical model used for hierarchical fault simulation.

a hierarchical model is referred to as hierarchical concurrent fault simulation. Another technique to fault simulate a hierarchal model is to use event driven serial fault simulation [144]. Symbolic fault simulation which is performed by evaluating Ordered Binary Decision Diagrams (OBDDs) is another technique which is used to perform the hierarchical technique [99]. The theory associated with OBDDs is presented in [37, 38]. Due to space limitations the theory associated with OBDDs is not described in this report. Concurrent fault simulation is preferred over serial fault simulation because concurrent fault simulation can compute all of the detectable faults for the DUT for a given input vector in a single simulation. Throughout the remainder of this report the term hierarchical fault simulation will be used to refer to hierarchical concurrent fault simulation since it is the most prevalent fault simulation technique for a hierarchical model.

A hierarchical fault simulator facilitates the development of test strategies during all phases of the design cycle. Specifically, test patterns can be developed and evaluated for each subdevice at each level of design abstraction [2, 61, 72, 144, 178, 179]. The test pattern generation strategy can be divided into two areas, detecting subdevice interconnect faults and detecting internal subdevice faults. The interconnect faults are commonly referred to as pin level faults since subdevices are often designed to be standalone ICs. Test patterns derived to detect pin-level faults are not as accurate as test patterns that are designed to detect all single stuck-at faults in a DUT [61]. In fact, Davidson has shown that pin-level faults are a subset of the total set of gate-level faults associated with a subdevice [61].

The ability to simulate internal subdevice faults at various levels of design abstractions typically requires a fault model associated with each level of design abstraction. Hierarchical fault simulation is capable of using fault models associated with any level of design abstraction. For example, the hierarchical fault simulator presented in [179] uses a functional fault model. In theory, hierarchical fault simulation can simultaneously use multiple fault models during a single simulation. The bookkeeping associated with the multiple fault model scenario is quite involved. One way to eliminate the bookkeeping is to have the faulty behavior of each device in the simulation use a fault library [47, 87, 159, 178, 179]. A fault library contains the correct and erroneous response of each device for all fault conditions associated with the device. Thus, the fault model for devices described at different levels of abstraction is essentially hidden with the use of fault libraries. Additionally, the use of fault libraries allows all devices in a simulation, regardless of the level of design abstraction, to be treated in an identical fashion.

Not all hierarchical fault simulators support multiple levels of fault model abstraction. Some hierarchical fault simulators require that the subdevice be represented at either the gate or switch level before internal faults can be inserted into the subdevice [72, 73, 144, 148]. Thus, this type of hierarchical fault simulator can only inject pin-level faults for subdevices which are represented at a high-level of abstraction such as functional or register transfer models.

One interesting feature which is possible with hierarchical fault simulation is dynamic model reconfiguration. Essentially, hierarchical fault simulation allows for a subdevice model at a higher level of abstraction to be replaced with a model of lower level of abstraction in a dynamic fashion. The replacement of one subdevice model with another in an automated fashion is referred to as dynamic model reconfiguration [148, 178]. An example of dynamic hierarchical model reconfiguration is included as Figure 5.11 to assist in illustrating this concept. The example diagram, Figure 5.11a, depicts a device which is comprises two subdevices, subdevice A and subdevice B. The outputs of the subdevice A are the inputs to subdevice B. The fault simulation technique begins by propagating the effects of all of the faults from the first device to the second device. For discussion purposes assume that both subdevices depicted in Figure 5.11a are represented by functional models and that gate-level models exist for each subdevice. The propagation simulation of the first device begins by determining if any input pin faults propagate to the output pins of the DUT. For the included example, the i_4 input is faulty and causes the output pin A_2 of subdevice A to be erroneous. The A_2 error causes the output pin o_1 of subdevice B to be erroneous. If output DUT errors are observed during fault simulation then the functional model for the evaluated sub-
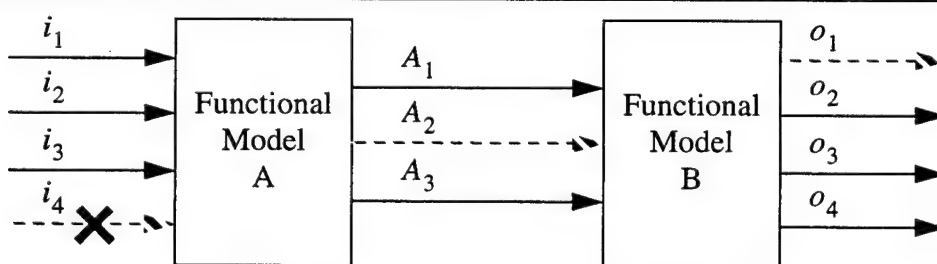
Figure 5.11a
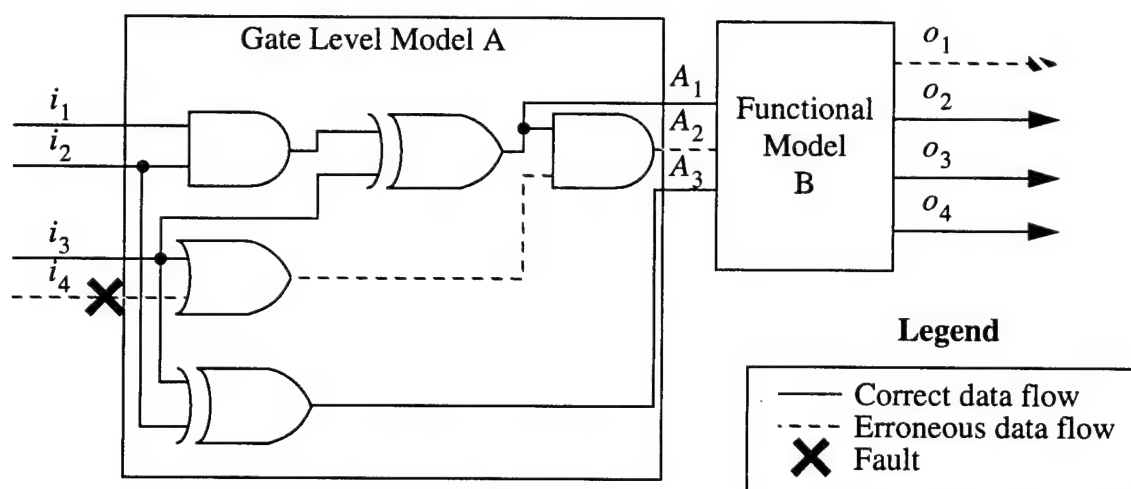
Figure 5.11b

Figure 5.11.   Dynamic hierarchical fault simulation example

device (subdevice A) is replaced with the gate-level model. The gate-level model is then fault simulated via the concurrent method to evaluate the gate-level fault list of the subdevice. The model replacement is performed to determine the internal gate-level signal faults which are detected for the given test vector. An example of the dynamic model reconfiguration is included as Figure 5.11b. The gate-level model has one internal fault which produces the same error effect as the i_4 fault.

Hierarchical fault simulation requires less computational resources than concurrent fault simulation. The computational cost of concurrent hierarchical fault simulation is $O\left(G\log_k\left(G\right)\right)$. The base of the log is defined as

$$G = k^l \tag{5.19}$$

where l is the number of levels in the DUT and k is a branching factor which conceptually represents the width of the DUT [87, 158, 178]. Thus, hierarchical fault simulation requires less computational resources than all other previously mentioned fault simulation techniques.

5.7. Circuit Structure Based Techniques

The primary goal of circuit abstraction methods is to decrease the amount of gate evaluations required for a single fault simulation. The reduction of computational complexity is achieved by exploiting the inherent structure present in the DUT. All of the known Circuit Structure Based (CSB) methods are applied to gate-level or switch-level models using the stuck-at or stuck-on/off fault models. Most CSB techniques exploit the fact that a circuit which contains no fanout can be evaluated via fault simulation as a linear function of the number of gates in the DUT $\left(O\left(G\right)\right)$ [18, 68]. Circuits which contain no fanout branches are referred to as fanout free circuits. The starting point for most CSB methods is to analyze the DUT to locate all of the Fanout Free Regions (FFRs) and the FanOut Stems (FOSs). An example circuit is included as Figure 5.12 to illustrate FFR and FOS.

The FOS in a circuit increases the fault simulation complexity significantly. The worst case complexity increase is on the order of $O\left(G^2\right)$. The circuit structure which provides the worst case complexity is a FOS which reconverges to the inputs of a single gate. This type of circuit structure is referred to as reconvergent fanout. An example of reconvergent fanout is provided in Figure 5.12. The key point to keep in mind is that only the portions of the circuit which are part of a reconvergent fanout region have the increase in complexity. Thus, if the FFR and FOS areas of the circuit are handled by separate fault simulation techniques the overall computational cost of simulating the entire DUT should decrease [16, 18, 26, 90, 98, 113, 126, 127, 134, 135, 136]. There are two basic approaches to exploiting the computational efficiency associated with FFR: (1) simulate the FFR to determine if the effect of FFR faults propagate to a FOS or Primary Output (PO) and then simulate the stem region if necessary [16, 18, 98, 126, 127, 134, 135, 136] and (2) simu-
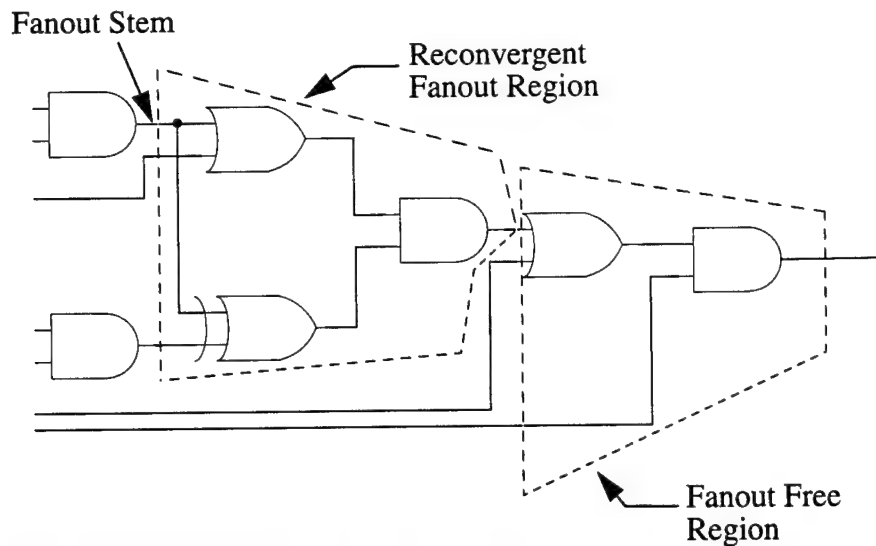
Figure 5.12. Circuit features exploited with CSB techniques.

late the FOS regions first and simulate the FFR if and only if an error from the FOS region propagates to the input of a FFR [90, 113, 187].

The fault simulation method to evaluate the FOS region is performed using any of the aforementioned fault simulation techniques; that is, serial, parallel, deductive, concurrent, differential, and hierarchical may be used. The methods to simulate the FFR in $O(G)$ time are limited to Critical Path Tracing (CPT) [6, 7] and event driven serial fault simulation. The CPT algorithm is described in Section 5.7.1. The details associated with serial fault simulation are provided at the beginning of Section 5. An overview of other existing CSB techniques is included as Section 5.7.2.

5.7.1. Critical Path Tracing

Critical path tracing consists of simulating the fault-free circuit and using the simulated signal values for tracing backwards from the Primary Outputs (POs) towards the Primary Inputs (PIs). The fault-free simulation is required in all fault simulations to determine the correct value of the POs. The correct PO values are then used to determine if injected faults are detected. Thus, the only additional processing step involved with CPT is the backward propagation phase. In subsequent material, a signal line l in the DUT for a given test pattern t has a critical value v if and only if t detects the fault l s-a-$\bar{v}$ [6, 7].

The first stage of the CPT algorithm is the fault-free simulation of the DUT. During the fault-free simulation the gate inputs which are sensitive are recorded. A gate input is considered sensitive if complementing the input causes the output of the gate to change value [6, 7]. For example, all inputs to an OR gate are sensitive if all inputs are 0. Conversely, if the output of an OR gate is 1 then an input is sensitive if and only if all other inputs are 0. The second phase of the CPT involves tracing the sensitive inputs backwards from the POs to the PIs. The tracing process begins by deter-

mining if the gate which drives a PO has any sensitive inputs. If the output gate has one or more sensitive inputs then the gate which set the sensitive input is evaluated to determine if it contains any sensitive inputs. The backwards traversal through the circuit continues until a PI is reached or a gate is reached which does not contain any sensitive inputs. An example circuit which is evaluated by the CPT process is included as Figure 5.13. The gate inputs which are sensitive are marked by a ● symbol. The back tracing begins at the output of gate 7 in Figure 5.13. The gate has a single sensitive input. Gate 6 which has two sensitive inputs sets the sensitive input to gate 8. Gates 1 and 2 set the value of the two sensitive inputs to gate 6. Gate 2 has one sensitive input which is a PI. At this point the CPT algorithm ends. The critical path traced by the CPT algorithm is indicated by bold signal lines in Figure 5.13. Each critical signal line l which has a value v detects an l s-a-$\bar{v}$ fault. [6, 7, 115].

The CPT technique locates all detectable stuck-at faults in the DUT if the DUT contains no reconvergent FOS. However, the CPT can be used with a DUT that contains reconvergent fanout gates. The CPT algorithm is considered to be an approximate algorithm for reconvergent fanout circuits because it does not locate all detectable faults in reconvergent fanout region. CPT does not detect the condition where multiple inputs of a reconvergent gate must be sensitized before a FOS fault is propagated to an input. A good illustration of a multiple sensitization requirement is included as Figure 5.14. In this example the output is set by a two-input OR gate whose fault-free input value is 11 (gate 4 in Figure 5.14). The inputs to gate 4 result from the FOS associated with
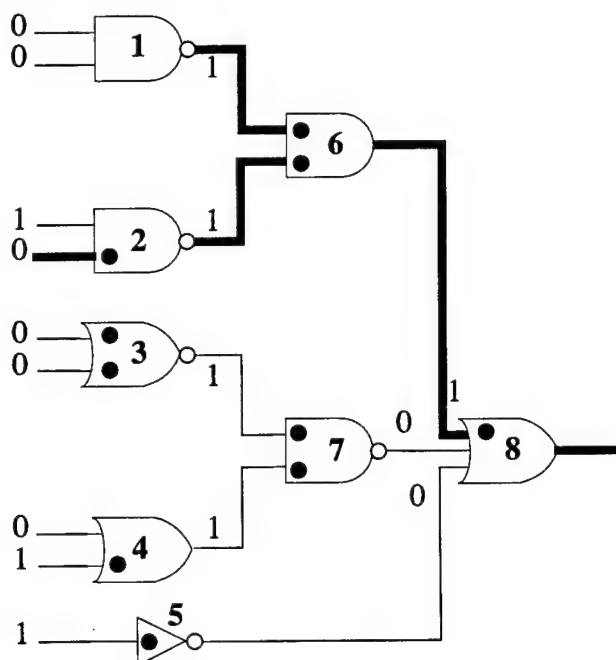
Figure 5.13. Critical path tracing example showing all salient features.
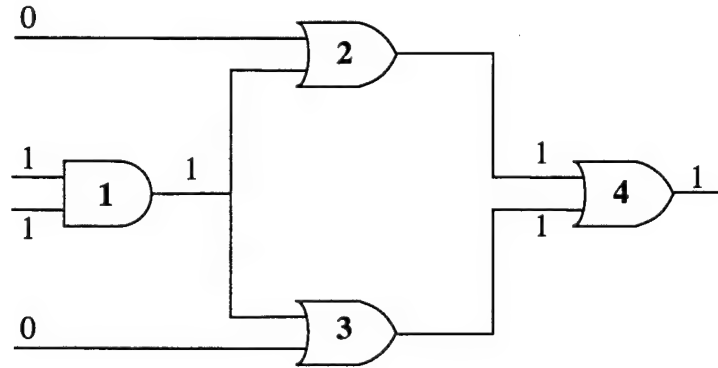
Figure 5.14. Fanout stem example where CPT fails to locate a fanout stem fault.

gate 1 reconverging. The outputs of gates 2 and 3 are controlled by the output of gate 1 so long as the other input to gates 2 and 3 are zero; that is, a 1 or 0 value at the output of gate 1 results in a 1 or 0 value on the outputs of gates 2 and 3, respectively. For discussion purposes assume that the FOS has a s-a-0 fault which affects the input to gates 2 and 3. The FOS fault causes the output of the circuit to produce an erroneous 0 output. This scenario is an example of a multiple input sensitization requirement that CPT does not handle. Specifically, neither input to gate 4 is a controlling input. The backward propagation phase of the CPT algorithm will stop at the output of gate 4 because gate 4 does not possess a controlling input. The inability to locate all faults in a reconvergent fanout region causes fault coverage estimates obtained by CPT to be pessimistic in nature [6, 7]. CPT is exact for all circuits which do not contain reconvergent fanout. Kitamura presents an exact critical path tracing technique that accounts for the multiple sensitized path scenario [115]. The downside to the exact CPT technique is that the computational cost is greater than $O\left(G\right)$.

Several extensions have been made to the original CPT algorithm. One technique is to use event driven PPSFP simulation to speed the evaluation of the DUT [17, 133, 205]. Likewise, Dalpasso presents a CPT technique for switch level models in [56]. The technique locates critical transistors based on the stuck-on/off fault model. A technique for using CPT on sequential synchronous circuits is described in [116, 141]. Additionally, Ramakrishnan extends the efficiency of CPT by using a more detailed analysis of the signals. CPT uses two categories of signal line classification, critical and noncritical. The Improved Critical Path Tracing (ICPT) uses six signal categories to improve the performance of the basic CPT algorithm. The six categories are: c_inhibitors, nc_inhibitors, non-inhibitor, propagator, absorber, and blocker. The purpose of the additional categories is to assist in determining if the FFR will propagate faults when an input is changed. Thus, ICPT provides a means to determine if a FFR will propagate faults when an input change occurs in a more computationally efficient manner than CPT [175].

5.7.2. Other Circuit Structure Based Techniques

There are several additional CSB methods which are used to exploit various circuit structure to minimize fault simulation computational cost. The additional CSB techniques can be divided into three categories: (1) determining fault sets that produce the same output error for a given input, (2) analysis of the FOS to determine error propagation behavior, and (3) determining the set of undetected nonlatent faults for a given input pattern. Each of these topics is described in the following paragraphs.

One technique for reducing fault simulation is to perform an analysis of the DUT to determine sets of faults which produce the same DUT output error for a given input. One accepted technique for determining the fault set information is to use fault collapsing. Roth in [181] introduced the concept of fault collapsing with the D algorithm. Conversely, Chen introduces a technique to perform fault collapsing on sequential synchronous DUTs [50]. Likewise, a Fault Information Tracing (FIT) technique that is used for fault simulation is presented in [116]. The FIT begins by simulating a fault-free DUT. Fault simulation is performed on sequential synchronous circuits by determining the faults which produce an error on a PO. The faults which produce an erroneous PO are determined in a fashion similar to CPT. The inputs to the gate which drives the PO are analyzed to determine which inputs produce the desired error. The input errors are then analyzed in a similar fashion as the erroneous output of the next level of gates in the DUT. The fault tracing ends when either a PI is reached or there exists no gate input fault which produces the desired error [116]. The faults associated with the corruption of a single PO belong to a fault set.

The second CSB method involves the analysis of the FOSs in the DUT. Specifically, if all gates attached to a FOS are unable to propagate a FOS error then there is no need to analyze the FFR attached to the FOS. One easy way to determine if the successor gates of a FOS can propagate an error is to analyze the set of inputs applied to each successor gate. Under certain inputs the error condition caused by the FOS will always be masked. Gates which exhibit this behavior are referred to as blocking gates [161]. For example, a two-input AND gate with a 00 input is a blocking gate for either input having a s-a-1 fault. A FOS which is attached to blocking gates prevents the propagation of a FOS signal error to a PO. Thus, a faulty FOS which is blocked is undetectable. All faults whose error must propagate through a blocked FOS to be detected are also blocked. Thus, the evaluation via fault simulation of the FFR which sets the FOS value is not required since all FFR faults are undetectable. The end result of the stem analysis is the location of DUT regions which contain undetectable, nonlatent faults for a given input vector. The undetectable faults are not evaluated by fault simulation for the current input vector. The stem analysis eliminates unnecessary fault simulations and therefore increases the efficiency of evaluating a fault list via fault simulation [161].

Another way to improve simulation efficiency is to eliminate all nonlatent faults which are undetectable for a given input vector from the list of faults to simulate. For a FFR, all nonlatent faults on nonsensitive gate inputs produce an error but are undetectable for the current input vector. Conceptually, this type of fault analysis is the inverse of the CPT. Akers [15] and Rudnick [183] present algorithms to locate a subset of the nonlatent undetectable faults for a given DUT and input vector. Both techniques rely on locating nonsensitive gate inputs to build the undetected fault list. The set of undetected faults identified by both algorithms is a subset of the total undetected fault set associated with the current input vector. Fault simulation is required to determine which remaining faults in the fault list are detected by the given input. Additionally, the technique presented in [15] is of linear computational complexity $(O(G))$.

5.8. State Initialization Simulation

There exists a class of synchronous sequential circuits where a state reset is not feasible. Testing this class of synchronous sequential circuits requires one to assume that the initial state of the DUT is unspecified. The use of three-value logic to place all memory elements in an initial X state is one common solution to this problem. However, the use of three-value logic with all states initialized to X results in overly pessimistic fault simulation results. Specifically, the fault coverage estimate of the uninitialized synchronous sequential circuit obtained using fault simulation with three-value logic is overly pessimistic [120, 121, 172]. There are several techniques available to increase the fault simulation accuracy. One technique entails comparing the output of the DUT after each input vector is applied. The multiple observation approach is referred to as Multiple Output Test (MOT). The use of MOT increases the coverage estimate but still results in a pessimistic coverage estimate [120, 121, 172]. The disadvantage of MOT is that it requires more output comparisons which increases the cost associated with testing a device.

Another approach to solve the state initialization problem is to use a multiple-pass simulation pass strategy. The first simulation pass uses traditional three-value logic with all state elements initialized to X. All faults detected during this first pass are dropped from the fault list. The second pass entails using partially specified states as the initial state of the DUT. Fault simulation is performed for each partially specified state to determine which faults are detected. For example, consider the case where the device under test has two flip flops. A set of partially specified states is given as $1X$ and $0X$. The third simulation pass involves complete state enumeration and evaluation. The evaluation of the complete state space can be performed in two ways: (1) enumerate all initial states in the DUT and perform fault simulation for each initial state, or (2) perform symbolic simulation. For a fault to be declared detected with the state enumeration approach the fault must be detected for all starting states [172]. The symbolic approach uses Ordered Binary Decision Diagrams (OBDDs) to symbolically simulate the faulty finite state machine [53, 120, 121]. Symbolic

simulation determines the output of the finite state machine for all possible starting states in a single simulation. The downside to symbolic simulation is that it is very memory intensive and as such only moderately sized circuits can be simulated with this approach [172]. The downside to the state enumeration approach is that state explosion can result in an inordinate number of initial states which results in an inordinate amount of simulation. Additionally, the low-level fault simulation technique employed by [120, 121, 172] is three-value serial fault simulation.

5.9. Hybrid Fault Simulation

Hybrid fault simulation combines multiple fault simulation techniques into a single fault simulator. The objective of hybrid fault simulation is to use the best features of each included technique to increase fault simulation efficiency. For example, the PROOFS fault simulator combines attributes from differential, serial, and parallel fault simulation techniques for the evaluation of sequential synchronous circuits [160]. The PROOFS fault simulation begins by simulating each gate in the DUT in an event driven fashion and storing each signal value. The initial fault-free simulation evaluates 32 good DUTs in parallel using the same input pattern. Also, the simulation uses four value logic; that is, 0, 1, X, and Z. The X and Z values represent unknown signal value and high impedance signal values, respectively.

Faults are inserted into the model in parallel. Thus, the PROOFS fault simulator uses a SPMFP technique. The effects of the inserted faults are propagated using an event driven simulation. All detected faults are dropped from the fault list. The state associated with each undetected fault is saved for the simulation of the next input vector. Like differential fault simulation, PROOFS does not restore the simulation to fault-free signal values before simulating the next set of 32 faults in parallel. Instead, the set of faults previously simulated is removed, the new set of 32 faults is applied, the state of the 32 faulty DUTs is restored to the appropriate value, and the DUT is simulated using an event driven approach. Once the entire set of faults in the fault list is evaluated then the next input vector is applied to the DUT. The state of the DUT is updated to contain the fault-free state calculated from the previous input vector. The DUT is simulated and the fault-free next state is saved along with the fault-free PO. Fault simulation begins by grouping 32 faults together, restoring the state associated with each fault to the DUT, and performing the event driven simulation. The detected faults are dropped from the fault list while the next state is saved for each undetected fault. The selection of 32 faults, restoration of the state associated with each fault, the event driven simulation, and the dropping of detected faults and saving the state of the undetected faults is continued until all faults in the fault list are evaluated. The evaluation of the remaining test patterns proceeds in a similar fashion [160].

The ordering of the fault list has a significant impact on the performance of PROOFS. The number of events to evaluate is minimized when adjacent faults in the fault list are similar. The

ordering requirement for the fault list is a byproduct of using a differential fault simulation approach in PROOFS. Like differential fault simulation PROOFS obtains the best performance when the fault list is ordered in a depth first organization starting at the POs and working towards the PIs [160].

5.10. VHDL-Based Fault Simulation

This section provides an overview on existing VHDL-based fault simulation techniques. There are essentially two categories of information provided in this overview: (1) techniques described in literature, and (2) commercial products. The following two subsections describe the literature and product surveys, respectively.

5.10.1. Literature Survey

Very little research has been focused on performing VHDL-based fault simulation. There are essentially seven different VHDL-based fault injection techniques described in the literature. Each of the seven techniques are described in the following paragraphs.

The first VHDL fault simulation approach uses an event driven SPMFP parallel fault simulation technique [156]. The authors note that the mapping of gate level evaluations to single machine instructions is lost when high-level data structures are used to store signal values. Since VHDL signals are represented by complex data structures during simulation, it is typically not possible to evaluate W parallel gates with a single host processor instruction. No data is presented to indicate the increase in efficiency which results in using VHDL SPMFP fault simulation versus VHDL serial fault simulation [156]. Additionally, this technique requires that the VHDL model be modified in a significant fashion. For this technique to be used in a tool the modification of the VHDL model must be automated. It is unclear the amount of effort which is required for the automated model modification.

The second approach adapts critical path tracing for use with VHDL gate-level models [194]. The major drawback of critical path tracing VHDL fault simulation is that a reverse propagation phase is required to determine the critical signal lines. The reverse propagation conceptually entails sending information from the outputs of the DUT to the inputs of the DUT; that is, information flows in the reverse direction as the normal gate level signal propagation [194]. Thus, all signal values in the DUT are bidirectional. The bidirectional information flow required of gate level signals is undesirable since the addition of bidirectionality can be viewed as changing the basic structure of the VHDL model.

The third technique entails the use of a behavioral VHDL fault model to derive the fault list of interest. The faults are injected into the DUT by modifying the VHDL source code [224]. There are two fundamental problems with this approach: (1) the compile time associated with each injected fault produces an unacceptable amount of overhead for large circuits, and (2) no attempt

is made to justify the completeness of the VHDL functional fault model. Specifically, no attempt is made to connect physical faults to the corruption of high-level behavioral VHDL statements. For this reason it is difficult to use this technique to estimate the test coverage of the device for stuck-at fault models.

The fourth approach involves translating the VHDL model to a C program which is then used to propagate the effect of a fault forwards and backwards through the DUT. A functional fault model is used to derive the fault list. The primary problem with this approach is that a VHDL simulation is not used to simulate the injected fault [168, 169, 170].

During the design of the fifth technique a theoretical analysis is performed to determine the fundamental fault injection methods for VHDL models. The authors state that there are three fundamental techniques for performing VHDL-based fault simulation: (1) using bus resolution functions to corrupt the signal value and a fault insertion process to control when the bus resolution function injects a fault (referred to as saboteurs), (2) modifying the VHDL source code to introduce faults (referred to as mutation), and (3) using simulator specific features to change the value of signals and variables during a simulation [110, 175]. Both the saboteur and mutation methods require the modification of VHDL source code and a recompile before fault simulation can occur. The recompile of the VHDL model is needed every time the location of the fault is moved. The overhead associated with the model recompilation is one major problem with this approach. The saboteur bus resolution function also introduces overhead. The amount of overhead caused by the bus resolution function is both simulator dependent and related to the amount of activity associated with the signal. The simulator specific fault insertion techniques do not require recompilation or any additional simulation overhead. However, to corrupt signals/variables with simulator specific features typically requires that the simulation of the DUT be halted, the desired fault condition is inserted, and the simulation restarted [110, 175]. Thus, simulator specific fault insertion has overhead associated with stopping and restarting the simulation to insert a fault. The other problem with simulator-based fault insertion is that the fault insertion is tied to a specific VHDL simulator.

The end result of the analysis is the development of the Multi-level Error/Fault Injection Simulation TOol (MEFISTO) [110]. The objective of MEFISTO is to provide a capability to perform fault grading during the design process. All three of the aforementioned fault insertion methods are used by MEFISTO. A 32-bit processor VHDL model is evaluated in [110] to demonstrate the feasibility of MEFISTO. The example fault simulation is performed on both a behavioral and structural VHDL model of a 32-bit microprocessor.

The sixth VHDL-based fault simulation technique relies on using a fault injection process to control a Bus Resolution Function (BRF) which injects a fault in a given signal [64]. One key advantage of this technique is that it can be used at any level of design abstraction. The fault injection technique allows for the fault simulation process to be readily automated so that the fault grad-

ing of a given DUT can be performed in a completely automated fashion. A second advantage of this approach is that the changes to the VHDL model required to insert the BRF and the controlling process can be performed in an automated fashion. The modified model is then recompiled once and any fault simulation where faults corrupt signals can then be performed. The presented fault simulation technique is simulator independent so any VHDL compliant simulator can be used. Another advantage is that the fault simulation technique allows for a mixed-level of modeling to exist in a given DUT. For example, consider the case where the DUT consists of 4 subcomponents. Assume that one subcomponent is represented by an algorithmic model which contains an approximate functional mapping but does not contain accurate timing information. The second component is modeled at a behavioral level and contains the correct functional mapping with approximate timing information, and the third component is a behavioral VHDL model before synthesis. The fourth component is represented by a structural VHDL model after synthesis. The presented fault simulation technique allows for all signal values in the example DUT to be corrupted by stuck-at faults starting at any time and lasting any duration. Thus, the technique presented by DeLong overcomes many of the limitations associated with the MEFISTO technique [64]. Specifically, the presented technique is simulator independent, requires only one model modification and recompilation cycle, and any signal in the model can be corrupted with a fault at any time for any duration.

The seventh VHDL fault simulation technique uses the concept of a super entity to insert faults into the VHDL model [44]. A graphical representation of a generic super entity is included as Figure 5.15 to facilitate discussion. A super entity is constructed by adding a data modulator to an existing VHDL entity. The purpose of the data modulator is to apply a mask vector to the output port of the super entity. The mask vector stores the type of fault corruption which is applied to the output port. The enable signal associated with the data modulator controls when the data modulator is active. The super entity approach allows for either permanent or transient faults to be inserted into the VHDL simulation [44].

The primary disadvantage with this technique is that the fault insertion technique only allows for the insertion of faults into the output port of a VHDL entity. Supporting only output port fault insertion is problematic because a designer may want to corrupt the internal signals contained in a given entity. If a given entity does not comprise low-level components then internal faults can only be inserted after the designer has redesigned the entity to incorporate low-level components. Having a fault simulation method which forces the designer to adhere to a given modeling methodology is undesirable. For example, consider the case where the designer wishes to procure commercially available models of components to speed the design process. If the commercial models do not adhere to the required modeling methodology then the designer is unable to fault simulate the design until the commercially available components are redesigned to adhere to the
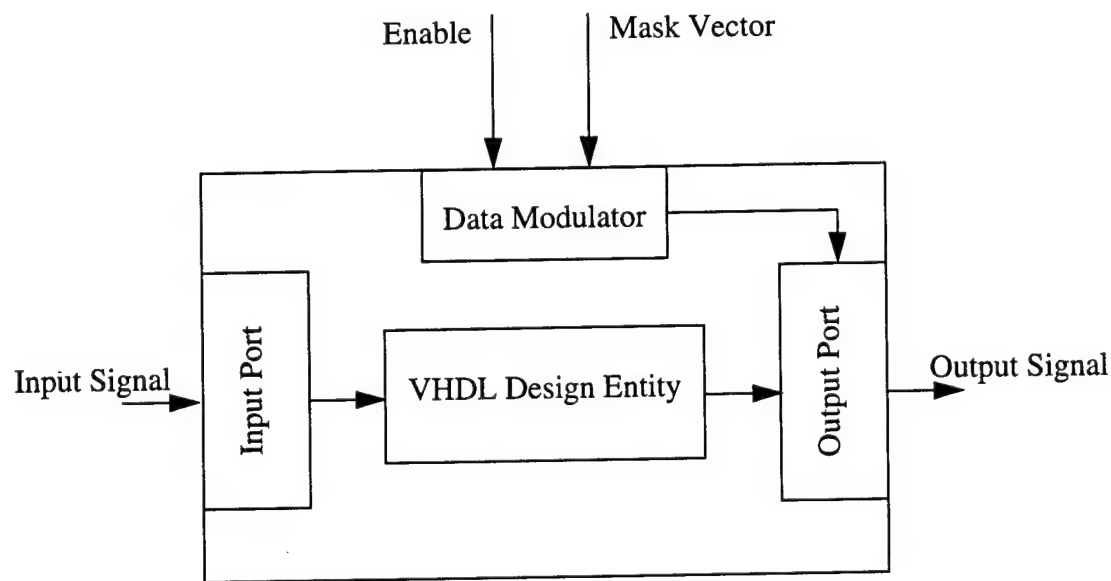
Figure 5.15. **Super entity fault insertion model.**

required methodology. Having the designer rework off the shelf models adds time back to the design cycle.

5.10.2. Commercially Available Products

There are a number of vendors that provide commercially available products to perform fault injection on gate level circuits described by a VHDL model. Specifically, Attest, Ikos, Synopsis, and ZyCAD produce fault simulators which accept VHDL gate level models [20, 104, 207, 236]. All four of the commercial products perform fault simulation using the same approach, concurrent fault simulation. The main drawback to the commercial products is that the fault simulation is not performed using a VHDL simulator. Each product uses the VHDL model as an input file format which defines the DUT. After the VHDL model is read then fault simulation is performed by executing a concurrent fault simulation program which is written in a high-level language such as C. The basic problem with this approach is that a VHDL simulator is not used to perform the fault simulation.

Another product that provides fault simulation information is VHDL Cover produced by TransEDA limited. VHDL Cover uses accepted software test paradigms to determine several software test coverage parameters. The basic idea is that a high-level VHDL model is essentially identical in structure to the source code of a high-level language program. The test strategies used for software should therefore be directly transferable to high-level VHDL models. The test metrics measured by VHDL Cover are: (1) percentage of statements evaluated, (2) percentage of branches

evaluated, (3) percentage of paths exercises, (4) percentage of processes triggered, and (5) percentage of state space evaluated. This type of information provides information on the completeness of the input set used to test the behavioral VHDL model. VHDL Cover also identifies redundant code which is never executed. The redundant code can then be removed from the model to eliminate unused hardware from the final design [212]. The main problem with this approach is that it is difficult to determine the fault coverage of the DUT with the information provided by VHDL Cover.

6. Parallel Processor Fault Simulation

One technique for increasing the performance of fault simulation is to use multiple processors to increase the available computational resources. Using multiple processors in this fashion is referred to as parallel processor fault simulation. The implementation of a given parallel processor fault simulation technique is influenced by three primary factors: (1) the architecture of the parallel processor host, (2) the uniprocessor fault simulation technique that is to be mapped to the parallel processor, and (3) the type of DUT to be evaluated. The types of host parallel processors available for fault simulation range from massively parallel machines such as the connection machine to supercomputers such as the Cray X-MP to a group of engineering workstations such as a cluster of SPARC[1] Workstations. All of the uniprocessor fault simulation techniques described in Section 5 can, in theory, be ported to a parallel processor environment. The third attribute which affects the implementation is the type of DUT to be evaluated. Specifically, the level of design abstraction, the type of fault model used, and whether the DUT is combinational, synchronous sequential, or asynchronous sequential machines affect the implementation of the parallel processor fault simulator.

This section provides an overview of existing parallel processor fault simulation techniques. The overview is organized based upon the architecture of the host parallel processor. For example, Section 6.1 provides an overview of parallel processor fault simulators which use vector based machines. Fault simulators which use massively parallel machines are described in Section 6.2. Likewise, fault simulators that require pipelined parallel processors are presented in Section 6.3. An overview of fault simulators that require loosely coupled parallel processor architectures is included in Section 6.4. Conversely, fault simulators that execute on a cluster of workstations are presented in Section 6.5.

6.1. Vector-Based Approaches

A vector machine is optimized to perform vector operations at a maximum rate. The optimization is performed by noting that the individual operations associated with a vector operation can be evaluated in an independent parallel fashion. For example, adding two n element arrays together

---

1. SPARC is a registered trademark of SPARC International, Inc.

requires n independent additions which can be performed in parallel. Vector-based fault simulation exploits the parallelism provided by the vector machine. The exploitation of the parallelism is best illustrated by example. For discussion purposes assume that the DUT is a combinational gate-level circuit. The DUT can be thought of as containing m levels of gates. A conceptual diagram of the jth level of a hypothetical combinational circuit is included as Figure 6.1. Since each gate can be evaluated independently then each gate in a given circuit level can be calculated in parallel. Bataineh presents a level oriented vector-based fault simulation approach which is simulated on a Cray Y-MP supercomputer. To fully exploit the Y-MP architecture, each 64-bit machine word is used to perform event driven SPMFP fault simulation. The 0 bit position stores the good value while bits 1-63 store the values associated with 63 faulty DUTs [25]. This technique is used to evaluate two-value zero-delay combinational circuit models. A similar technique is presented in [152] to fault simulate two-value zero-delay synchronous sequential on a vector-based supercomputer.
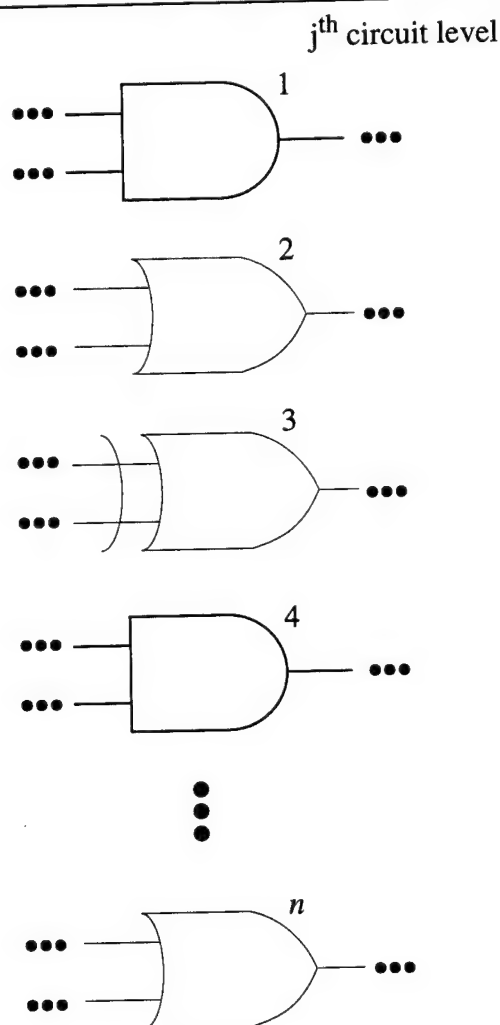
Figure 6.1. The jth level of a hypothetical combinational circuit.

Vector machines are also well suited to event driven PPSFP fault simulation. The one difficulty with vector-based PPSFP fault simulation is that the efficiency of the simulator drops after all the easy to detect faults have been detected. The remaining undetected faults are typically referred to as hard to detect faults or random pattern resistant faults. The reason for the drop in efficiency is that hard to detect faults have a high probability of either remaining latent or not propagating very far before the effect of the fault is masked. One way to overcome this limitation is to combine the SPMFP technique with the PPSFP technique to produce a Parallel Pattern Multiple Fault Propagation (PPMFP) method. The objective of PPMFP is to increase the likelihood of detecting at least one hard to detect fault when evaluated with parallel input patterns. Thus, PPMFP evaluates parallel patterns for multiple faulty circuits in one fault simulation pass. Each faulty circuit contains one single stuck-at fault. As the size of the fault list decreases with PPMFP the number of parallel patterns is decreased while the number of multiple faulty circuits is increased. A Cray X-MP [59] and a Fujitsu FACOM VP-200 [105, 106] supercomputer have been used to execute the PPMFP algorithm.

6.2. Massively Parallel Processor Based Approaches

A variety of approaches have been used to perform fault simulation on massively parallel processors. A massively parallel machine is defined to be a multiprocessor which contains more than 100 Processing Elements (PEs). A connection machine with 65,536 PEs [153, 154] and an IBM RP3 with 512 PEs [154] have been used to perform fault simulation. One technique for performing massively parallel processor fault simulation is to map each gate in the DUT to a PE. To minimize communication overhead, the output of each gate is also assigned a PE to propagate the output signal to the appropriate PEs. Each level of the circuit is evaluated in parallel. Likewise, each gate evaluation is performed using PPSFP fault simulation [153, 154, 155]

Another optimization step is achieved by noting that the next set of parallel patterns can begin evaluation before the POs are calculated for the current input vector. Evaluations performed in this fashion are equivalent to changing the inputs after all the gates which are attached to the inputs are evaluated but before the POs are set. The aforementioned evaluation process is analogous to a pipeline instruction execution architecture on a microprocessor. For this reason this type of input evaluation is referred to as pipelined simulation. The objective of pipelining is to minimize the amount of time required to evaluate the DUT for multiple parallel patterns. If the DUT contains n circuit levels then the earliest that two parallel patterns can be evaluated without pipelining is the time required to evaluate $2n$ circuit levels. With pipelining, the number of level evaluations is typically less than $2n$.

The key attribute associated with this type of pipelined evaluation is determining the number of simulation level evaluations which must be performed to assure the correct output is produced,

at least momentarily, regardless of whether the input is changed. The minimum number of gate evaluations is determined by analyzing the structure of the DUT. The purpose of the analysis is to determine the minimum and maximum propagation paths through the DUT. For the levelized evaluation approach the propagation path timing is calculated by simply counting the number of gates that an input signal must traverse before setting an output. The maximum and minimum delays are referred to as l_{max} and l_{min}, respectively. The maximum time that it takes a signal to propagate from the output of one gate to the input of another gate is $l_{max} - l_{min}$. Thus, the next input pattern can be evaluated after $l_{max} - l_{min} + 1$ gate levels are evaluated. The optimum amount of pipelining occurs when $l_{max} = l_{min}$ [153, 154, 155]. A circuit example is included as Figure 6.2 to assist in illustrating this concept. The maximum gate delay for this circuit is five while the minimum delay is two. Thus, for this example $l_{max} = 5$ and $l_{min} = 2$. The next input pattern can be applied to this circuit after 4 gate levels have been evaluated. Using pipelining to evaluate this example circuit eliminates the time associated with one gate simulation for the evaluation of each test pattern applied after the first test pattern. The first test pattern evaluation requires the simulation time required for five gate levels.

Another fault simulation approach is to assign a PE to each fault and one PE for the good circuit. Each PE then performs PPSFP fault simulation. This approach requires that the parallel machine contain $F + 1$ PEs for all faults to be evaluated in a single pass. The result of each PE which performs PPSFP is collected and the detected faults are then dropped from the fault list. Only the PEs which contain undetected faults perform PPSFP fault simulation for the next set of

$$l_{max} = 5$$
$$l_{min} = 2$$
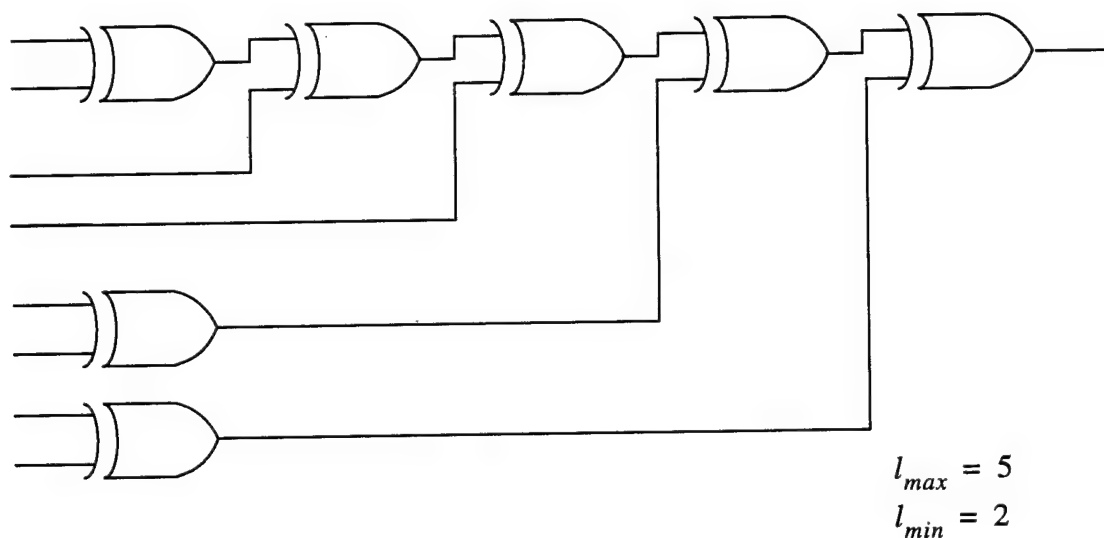
Figure 6.2. Example circuit depicting maximum and minimum delay calculation.

input patterns. The process is continued until all input vectors are evaluated or all faults are detected [153, 155, 162].

6.3. Pipelined Parallel Processor Techniques

The use of pipelining is one technique which is employed to map a uniprocessor fault simulation algorithm to a parallel processor environment. There are two fundamental techniques employed to incorporate pipelining into a fault simulation technique: (1) exploit the structure of the DUT, and (2) pipeline the sequential steps associated with a uniprocessor fault simulation algorithm. The following paragraphs describe the two fundamental approaches.

The first pipelining method described involves exploiting the structure of the DUT. The first step involved is the analysis of the DUT to create a Task Precedence Graph (TPG). A TPG determines the set of components in the DUT which can be evaluated independently. Specifically, each task maps to a given level of the DUT. Figure 6.1 provides an example DUT circuit level. Once the TPG is created then each task is assigned to a specific PE. Thus, the depth of the pipeline is determined by the number of tasks in the TPG. The fault simulation strategy employed at each stage of the pipeline for the technique presented in [165] is SPMFP parallel fault simulation. However, Li maps the DUT into a cellular automata paradigm to determine the independent levels of the pipeline [131]. The fault simulation is performed on each stage of the pipeline using a PPSFP fault simulation approach [131].

The second pipelining technique involves mapping the sequential steps associated with each component simulation to a sequence of steps that fits the pipeline paradigm. Typically, this category of pipelined fault simulation is applicable to uniprocessor fault simulation techniques which require a large number of processing steps to evaluate a component. Deductive and concurrent fault simulation are two uniprocessor techniques that require a large number of steps to evaluate a single component. Specifically, both concurrent and deductive use computationally intensive mechanisms to propagate the fault list for a given component. For example, in concurrent fault list propagation the following five major steps are performed: (1) calculate the correct component output, (2) add the propagated input fault list to the component fault list, (3) add the component faults which may produce an incorrect output to the fault list, (4) prune the fault list to contain only faults which produce a component output error, and (5) produce the output signal which contains the correct output and pruned component fault list. At a minimum, the concurrent fault list algorithm can use a 5-stage pipeline to optimize the fault simulation of each component. A technique developed by Stein uses a 5-stage pipelined approach [206]. Further low-level analysis of a given concurrent fault simulator implementation can result in a longer pipeline. Bose [31] and Agrawal [8] present pipelined concurrent fault simulation which uses a 14 and 12 stage pipeline, respectively.

6.4. Distributed Parallel Processors

A parallel processor architecture is considered distributed if there is a large time penalty associated with interprocessor communication. The design of a fault simulator which executes on a distributed parallel processor should minimize the amount of interprocessor communication to maximize fault simulation performance. One approach is to partition the DUT into clusters of components. Once the inputs are determined for each cluster then fault simulation can be performed on the cluster independent of all other clusters. One partitioning strategy is to group m levels of the DUT together to form circuit regions. Each circuit region is mutually exclusive of all other circuit regions. Each circuit region is then divided into p clusters each containing approximately the same number of components. The cluster partitioning is performed such that each cluster's inputs do not depend on the output of any other cluster in a circuit region. Thus, each cluster associated with a circuit region can be evaluated independently and in parallel. The structure of the DUT has a significant impact on the ability to produce independent partitions which can be fault simulated in parallel.

One way to achieve independent partitioning is to analyze the DUT and locate regions in the circuit that possess no interconnection. Conceptually, regions which are not interconnected are equivalent to sets that are mutually exclusive. Huisman presents a mechanism for determining mutually exclusive regions in the DUT based on a concept referred to as downcone independence [100, 101, 102]. A downcone is a circuit region which contains multiple inputs which are evaluated to produce one or more outputs. A downcone also contains no feedback loops. The technique presented in [100, 101, 102] partitions the circuit based on fan-in cones which drive latches. The DUT is partitioned in an automated fashion during a preprocessing step. Levendel [130] also presents a partitioning technique to divide the DUT into independent segments which span the depth of the DUT. Unfortunately, very little information is provided in [130] to describe how the partitioning process is performed. Conversely, the technique presented in [76] partitions the circuit by hand. Each cluster is then fault simulated using either the concurrent fault simulation technique [76, 100, 101, 102, 150] or the event driven SPMFP parallel fault simulation technique [130].

Another approach for parallelization of fault simulation is based on partitioning the fault list. Each fault list portion is assigned to a unique processor and a uniprocessor fault simulation algorithm is executed independently on each processor. Goel [81] and Motohara [149] present a technique which partitions the fault list among a pool of PEs. Each PE performs concurrent fault simulation to determine the fault detection status of each fault associated with a PE's fault list.

One of the key attributes for maintaining optimum efficiency on a distributed parallel machine is assuring that each processor is busy doing useful work for the entire fault simulation. Assuring that all processors are not idle is referred to as load balancing. The optimum technique used for

load balancing consists of two parts: (1) a static assignment of clusters to PEs, and (2) a dynamic reassignment of clusters if a PE becomes idle during the fault simulation. The static assignment is done prior to fault simulation, while the dynamic reconfiguration is done during fault simulation. The distributed processor fault simulation technique described in [100, 101] supports the static and dynamic features of optimum load balancing. Conversely, the technique presented by Ghosh relies on static partitioning to achieve load balancing [76].

6.5. Parallel Workstation Based Approaches

One technique for increasing the efficiency of fault simulation is to use a cluster of workstations to perform the fault simulation is parallel. This type of fault simulation is referred to as parallel workstation fault simulation. The use of parallel workstations in this fashion also fits the distributed parallel processor paradigm. However, the parallel workstation approach is placed in a separate category because most engineering design environments already contain a large number of workstations. For this reason, it is possible to implement a parallel fault simulation technique without having to procure an expensive multiprocessing platform; that is, simply use the existing cluster of workstations.

There are two basic strategies used to parallelize a uniprocessor fault simulation algorithm on a cluster of workstations. The first approach involves partitioning the fault list into n groups and running a uniprocessor fault simulation on n workstations independently. Markas presents an event driven SPMFP fault simulation approach which partitions the fault list into n groups [139]. Each workstation is given a fault list partition to simulate independently of all other workstations. The partitioning of the fault list has a dramatic effect on the performance of the fault simulation. Three different static partitioning schemes were explored in [139] including random, depth first, and topological cluster. The random approach randomly sampled the fault list without replacement to generate n fault partitions. The depth first technique orders the fault list from PO to PI. The ordered fault list is then partitioned into n fault lists. The clustered technique divides the fault list into topological clusters, and this technique provides the best performance. Also, a dynamic reconfiguration technique is used to move faults from a busy workstation to an idle workstation [139].

The second approach used to parallelize a uniprocessor fault simulation involves exploiting the hierarchy of the DUT. A technique for simulating multiple representations of a DUT in parallel using hierarchical concurrent fault simulation is given in [66]. The basic approach is to create multiple representations of the DUT with different subdevices represented at different levels of abstraction. Specifically, a DUT model which incorporates a gate-level model for only one subdevice with all other subdevices represented at a high-level of abstraction is created. Each DUT model is evaluated to determine if the faults contained in the gate-level portion of the DUT are detected by the set of input vectors. Each unique representation is given an identification number.

A total of n unique representations are evaluated in parallel on n workstations. Whenever a workstation completes the fault simulation for a given representation the results are saved and another unevaluated DUT representation is selected for evaluation. This process is continued until all DUT representations are evaluated for all input vectors [66].

7. Hardware Accelerator Fault Simulation

Another technique for reducing the fault simulation time associated with a given DUT is to incorporate a hardware accelerator into the fault simulator. A hardware accelerator is a specialized device whose purpose is to reduce the simulation time of a component model during the design process. A hardware accelerator is typically attached to an engineering workstation that is used to design the DUT. The simulation of a device on the hardware accelerator begins by compiling the device model into a format that is appropriate for the hardware accelerator. The compiled model is then transferred to the hardware accelerator. The hardware accelerator simulates the model and transfers results back to the workstation. Often, the time required to compile a model in conjunction with the transfer time of data to and from the hardware accelerator is greater than the time required for the hardware accelerator to simulate the model. In fact, the compile and transfer time can often negate any simulation speedup provided by the hardware accelerator [146].

The internal structure of a hardware accelerator typically contains multiple PEs with local memory and a high speed global interconnect mechanism. For example, the Simulation Processor (SP) contains 64 PEs each of which can simulate 64 K of primitives. Each PE is connected to a crossbar switch. The SP hardware accelerator is an event driven gate-level simulation engine [97]. The Yorktown Simulation Engine (YSE) is another hardware accelerator which contains 25 PEs which are interconnected by a crossbar switch. Each PE can simulate up to 8 K of gates. The YSE is a gate-level simulator [65, 166, 167]. The Microprogrammable Accelerator for Rapid Simulations (MARS) is another accelerator which can contain up to 256 clusters. Each MARS cluster can contain up to 16 PEs. The PEs in a cluster are connected via a 16 X 16 crossbar switch. Each cluster is connected to a binary 8-cube global interconnection network. Additionally, MARS is an event driven gate and transistor level simulator [9]. More detail on the historical evolution of hardware fault simulators and the low-level details associated with the internal structure of the hardware simulators is provided in [30]. Likewise, an overview of the use of hardware accelerators in the design process along with a list of commercially available hardware simulators is described in [146].

Hardware accelerators can be used with any fault simulation technique that maps directly to some type of hardware model. Unfortunately concurrent, deductive, and hierarchical fault simulation techniques all require a fault list propagation technique that does not map directly to a static hardware model. Typically, hardware accelerators are used to speed the evaluation of event driven parallel fault simulation [3].

8. Fault Grading Techniques

Fault simulation of the DUT provides information concerning the quality of the test pattern set for a given DUT. Specifically, simulating the entire fault set for a given test pattern set determines which faults are detected by the test pattern set and which are not detected. The fault detection data is then used to determine the percentage of detected faults when the test pattern set is applied to the DUT. The percentage of detected faults is referred to as fault coverage and the process of estimating fault coverage is referred to as fault grading.

There are several different accepted fault grading methods. An overview of the different fault grading techniques is provided in this section. One feature which is commonly used during the fault grading process is the concept of fault equivalence in the DUT. An overview of fault equivalence as it relates to fault grading is provided in Section 8.1. The various fault grading techniques are then introduced. Traditional fault grading techniques are described in Section 8.2. Likewise, the fault grading technique used by the United States Department Of Defense (DOD) is defined in MIL-STD 883 and an overview of this method is provided in Section 8.3. The relationship between fault coverage and test quality is described in Section 8.4. Finally, nontraditional fault grading methods which do not use traditional fault simulation data are described in Section 8.5.

8.1. Fault Equivalence

One technique which is commonly used to reduce the amount of fault simulation required is to exploit the fault equivalence of the DUT. The type of fault equivalence commonly used in fault grading is referred to as device-level fault equivalence. The set of device-level equivalent faults is derived from the fault model used to construct the fault list and the structure of the device itself. The concept of device-level fault equivalency is best illustrated by example. As a starting point consider an n-input AND gate as the device and the stuck-at fault model is used to evaluate the DUT. The fault list for the n-input AND consists of two stuck-at faults for each input and two stuck-at faults for the output. For this particular example, a single stuck-at 0 fault on the output of the AND gate is equivalent to a single stuck-at 0 fault on any input to the AND gate. For this reason the stuck-at 0 output fault on the AND gate can be eliminated from the fault list. Further analysis can be performed for each type of device to determine the equivalent faults. The information provided using device-level fault equivalency can be used during the fault grading process. An overview of the traditional fault grading approach is provided in the next subsection.

8.2. Traditional Fault Grading Approaches

The traditional fault grading methodology can be divided into two broad categories, exhaustive fault simulation techniques and fault sampling with statistical estimation of fault coverage approaches. The exhaustive method requires that all faults in the fault list of the DUT be evaluated

using fault simulation/fault equivalency techniques. The fault coverage is estimated from the knowledge of the complete fault set.

The derivation for the coverage estimate begins by noting that fault coverage is a percentage which is given as

$$C = Per(D, F) \tag{8.1}$$

where D is the fault detection event, F is the fault space of the DUT, and $Per()$ determines the percentage of detected faults over the entire fault space F. The fault grading derivation continues by showing how the detection event associated with the individual faults contribute to C. The following expression is used to continue the derivation

$$Per(D, F) = \frac{1}{n} \sum_{i=1}^{n} P(D_i) \tag{8.2}$$

where $P(D_i)$ is the detection probability of the ith fault in the fault space F and n is the number of faults in the fault space.

The results provided by the fault simulation/fault equivalency analysis measures the detection probabilities for each fault in the fault list. The measured data is binomial in nature; that is, a detected fault has a conditional probability of 1.0 and an undetected fault has conditional probability of 0.0. Stating this concept in mathematical terms

$$P(D_i) = \begin{cases} 1 \, \forall \text{detected} \\ 0 \, \forall \text{undetected} \end{cases} \tag{8.3}$$

With the knowledge gained from the fault simulation/fault equivalence analysis Equation (8.2) can be simplified to

$$C = Per(D, F) = \frac{1}{n} \sum_{i=1}^{n} P(D_i) = \frac{n_d}{n} \tag{8.4}$$

where n_d is the number of detected faults.

The exhaustive fault grading process described previously ignores one fundamental issue which causes fault grading to be problematic for some DUT. Specifically, the occurrence of redundant faults was ignored during the previous derivation. A redundant fault is a fault that can not be detected if the entire input space of the DUT is evaluated. Typically, a redundant fault is caused when the input or output of a redundant gate in the DUT is faulty. The most common reason to leave redundant gates in a circuit is to eliminate static and dynamic hazards from the circuit [140]. For a designer to claim that a fault is redundant then the designer must show that the fault can not produce an erroneous output for any input applied to the DUT. Typically, a designer uses an DATPG program which exhaustively searches the input space of the DUT to prove that a given fault is redundant.

The only modification required to incorporate redundant faults in the aforementioned fault grading process is redefining n in Equation (8.4). Essentially, the total fault set is reduced by eliminating the redundant faults. The removal of redundant faults is described mathematically as

$$n = n_T - n_{redundant} \qquad (8.5)$$

where n_T is the number of faults contained in the complete fault list and $n_{redundant}$ is the number of redundant faults contained in the total fault list. Equation (8.4) is then evaluated on the reduced fault set which contains no redundant faults.

The second fault grading technique is the use of random sampling to obtain a statistical estimate of the fault coverage associated with an input vector set for a given DUT. The basic premise is that evaluating the entire fault set to estimate the fault coverage is too costly in time and computer resources. Thus, a small random sample of faults is selected and evaluated. A coverage point estimate is calculated from the fault simulation results. One typical point estimate is given as

$$\hat{C} = \frac{n_d}{n} \qquad (8.6)$$

where $\hat{C}$ is the point estimate, n is the number of samples, and n_d is the number of sampled faults which are detected. Traditional statistical analysis is performed to calculate a confidence interval at a given confidence level [75]. Either a double-sided [11] or single-sided [232] confidence interval is calculated. The double-sided confidence interval calculation is derived from the central limit theorem. Conversely, the single-sided confidence interval is derived from a Bernoulli distribution which is approximated by a Poisson distribution. The end result of the confidence interval calculation is a statistically valid estimate of fault coverage which is obtained in a computationally efficient fashion.

Typically, a reduced fault list is used when statistical fault grading is employed. The reduced fault list contains only one fault from each device-level equivalent fault class. Also, all redundant faults are eliminated from the reduced fault set. Sampling the reduced fault set in a random fashion provides the best representative sample of the fault population. Thus, using the reduced fault set provides the most accurate statistical fault coverage estimate.

8.3. MIL-STD 883D

A group of fault grading techniques are described in MIL-STD 883D method 5012 [145]. The first step in the MIL-STD fault grading process is partitioning the DUT into m regions. Each region falls in one of two categories, Gate-logic (G-logic) partitions and Block-logic (B-logic) partitions. A G-logic partition consists entirely of logic gate components which are connected by signal lines. Conversely, a B-logic partition contains functional elements such as RAM, ROM, and ALU. The fault model associated with G-logic is the single stuck-at fault model. B-logic partitions, on the other hand, use a designer specified test strategy. The designer is responsible for supplying the

faults necessary to evaluate the test strategy. The sufficiency of the selected test strategy and the faults used to evaluate the test strategy must be justified by the designer.

The fault coverage estimation process begins by constructing the fault list for each DUT partition. The fault list for each G-block is constructed by selecting one fault from each device-level equivalent fault class. MIL-STD 883D provides a list of representative faults for each equivalence class associated with a specific logic gate type as shown in Table 8.1. The fault list associated with each B-logic partition is constructed to fully exercise the test algorithm selected by the designer.

Table 8.1. Representative faults for the equivalence classes [145].

Stuck-at Faults	Type of logic line in logic model
s-a-1	Every input of a multiple-input AND or NAND gate
s-a-0	Every input of multiple-input OR or NOR gate
s-a-0, s-a-1	Every input of multiple-input components that are not AND, OR, NAND, or NOR gate
s-a-0, s-a-1	Every logic line that is a fan-out origin
s-a-0, s-a-1	Every logic line that is a primary output

The fault grading process associated with each DUT block is accomplished by one of three techniques. The first method is analogous to the exhaustive fault grading technique described in Section 8.2. The fault coverage is estimated by first determining the detection status of each fault in the fault list. The collected fault detection data is then used to evaluate

$$C = \frac{n_d}{n} \tag{8.7}$$

where n is the number of faults contained in the fault list and n_d is the number of detected faults. A fault associated with a partition is considered to be detected if the fault causes an error to occur at some output or test point on the DUT. Equation (8.7) is used to estimate coverage for both G-blocks and B-blocks.

The designer is allowed to eliminate undetectable (redundant) faults from the fault list. The burden of proof is on the designer to demonstrate that a given fault does not produce a primary output error for all possible input vectors applied to the DUT before the fault can be categorized as undetectable.

The second acceptable fault grading process is to randomly sample n faults from the fault list. The designer specifies and justifies the distribution used to generate the set of sampled faults. The n selected faults are then evaluated via fault simulation. A point estimate for the fault coverage estimate is calculated by evaluating

$$\hat{C} = \frac{n_d}{n} \tag{8.8}$$

A confidence interval based on a 95% confidence level is calculated. MIL-STD 883D provides a table containing the penalty parameter r and is duplicated in this document as Table 8.2. The penalty parameter is subtracted from the coverage point estimate to determine the lower side of the coverage confidence interval. Performing this operation provides

$$\hat{C}_{low} = \frac{n_d}{n} - r \tag{8.9}$$

where r is a penalty parameter and $\hat{C}_{low}$ is the lower side of the confidence interval. The lower side of the confidence interval is used as the coverage estimate since it provides the most conservative estimate at a 95% confidence level.

Table 8.2. Fault coverage lower bound sample size using procedure 2.

r	n
0.010	6,860
0.015	3,070
0.020	1,740
0.030	790
0.040	450
0.050	290

The third accepted technique for fault grading technique is to first determine the minimum coverage value which is acceptable. The designer then refers to a table to determine the minimum number of fault simulations to show that the lower side of a confidence interval with 95% confidence is achieved assuming that all simulated faults are detected. The table is reproduced in this document and is included as Table 8.3. For example, if a 90% coverage estimate is desired then a minimum of 29 randomly selected faults must be simulated and must be detected.

Once each G-block and B-block is fault graded by one of the three methods then the overall fault coverage estimate for the DUT is calculated. The calculation begins by estimating the transistor fraction associated with each G-logic and B-logic partition. The coverage estimate for the DUT is obtained by summing the product of the transistor fraction and the coverage estimate for each partition. Performing this operation provides

$$C = \sum_{i=1}^{m} C_i T_i \tag{8.10}$$

Table 8.3. Fault coverage lower bound sample size using procedure 3.

F	n
50.0%	5
55.0%	6
60.0%	6
65.0%	7
70.0%	9
75.0%	11
76.0%	11
77.0%	12
78.0%	13
79.0%	13
80.0%	14
81.0%	15
82.0%	16
83.0%	17
84.0%	18
85.0%	19
86.0%	20
87.0%	22
88.0%	24
89.0%	26
90.0%	29
91.0%	32
92.0%	36
93.0%	42
94.0%	49
95.0%	59
96.0%	74
97.0%	99
98.0%	149
99.0%	299

where T_i is the per unit transistor area associated with the ith partition and C_i is the coverage estimate associated with the ith partition.

8.4. Test Quality as a Function of Test Coverage

The objective of fault grading is to determine the quality of the test pattern set to detect all modeled faults in the DUT. The fault grading process provides a quantitative method to measure test pattern quality. The estimated fault coverage value also provides a qualitative measure on the quality of the manufactured DUT. The ultimate goal of any testing strategy is to increase the quality of the manufactured DUT by eliminating defective components. The quality of the DUT is a function of two parameters: (1) fault coverage and (2) manufacturing yield. Both parameters are probabilities. The fault coverage estimate obtained via fault grading is converted from a percentage to a probability for the test quality calculations. Three assumptions are required for the percentages to probability conversion: (1) the fault model used represents all faults that the DUT can encounter, (2) all faults in the fault list are equally likely to occur, and (3) the fault occurrence events are independent.

The yield of a manufacturing process is the probability of a given DUT being fault-free for a given part population. A yield of 1.00 indicates that the manufacturing process is perfect and no defective parts are manufactured. Likewise a yield of 0.1 indicates one part in ten is fault-free. The yield of a manufacturing process can be estimated either through statistical estimation or analytical modeling [111]. Yield and test coverage are used as parameters to determine test quality. Conceptually, this relationship is given as

$$\xi = F(C, \phi) \tag{8.11}$$

where ξ is the quality of the manufactured device, C is test coverage, and ϕ is the yield. A simplistic quality measure is derived by noting that for a defective part to be sent to the field requires that a faulty part must erroneously pass the testing methodology [222]. The probability of a faulty part being manufactured is given as

$$P(F) = (1 - \phi) \tag{8.12}$$

The probability of not detecting a faulty device is

$$P(U|F) = 1 - P(D|F) = 1 - C \tag{8.13}$$

where $P(U|F)$ is the conditional probability of not detecting a fault given there is a fault in the DUT, and $P(D|F)$ is the conditional probability of detecting a fault given that there is a fault in the DUT. The probability of having a faulty component not be detected is

$$(1 - C)(1 - \phi) = 1 - \xi \tag{8.14}$$

Rearranging Equation (8.14) and solving for ξ provides

$$\xi = 1 - (1 - C)(1 - \phi) \tag{8.15}$$

There are numerous quality models which use fault coverage and yield as parameters [12, 54, 191, 210, 222, 233]. The analytical calculation of quality has one major problem which limits its usefulness; that is, the yield of the manufacturing process must be known. Unfortunately, it is difficult if not impossible to obtain yield information from a manufacturer. Companies typically consider the manufacturing yield information to be proprietary in nature. If the yield of a manufacturing process is known then it is relatively straight forward to determine the actual profitability of the manufacturing process. Having individuals external from a company know the profitability of the company's manufacturing process places a company in a bad position. Thus, the quality models can be used as an metric by the manufacturer of the DUT. For this reason, quality models have found limited use.

8.5. Other Nontraditional Fault Grading Techniques

All of the fault simulation techniques described in the previous sections describe exact methods; that is, fault simulation techniques which present exact results. While fault simulation can be used to estimate various DUT criteria, this report focuses on using fault simulation to estimate coverage. The coverage estimate obtained via exact fault simulation is accurate so long as the fault model used represents all faults that the DUT can encounter. Given the three criteria are satisfied then coverage estimation via exact fault simulation can be considered an accurate estimate of the probability that a fault will be detected in the DUT for the given input test pattern set.

The downside of exact fault simulation is the high computational cost associated with evaluating complex DUTs. One technique for overcoming this limitation is to use probabilistic fault evaluation techniques. The end goal of probabilistic methods is to provide an approximate coverage estimate which provides the desired level of accuracy at a significantly lower computational cost than exact fault simulation. Approximate fault simulation techniques use probabilistic fault detection techniques.

Probabilistic fault detection uses the concepts of controllability and observability to estimate the probability of detecting a stuck-at fault on a signal line. The controllability of a signal l represents the ability of a given input vector to set l to a value v. Conversely, the observability of a signal l represents the ability of a PO to determine that l is set to a value v [85]. Probabilistic fault detection places a probability on the controllability and observability of each signal line contained in the DUT. For sake of discussion the one and zero controllability probability of line l is referred to as $C_1(l)$ and $C_0(l)$, respectively. Likewise the one and zero observability probability is denoted by $O_1(l)$ and $O_0(l)$.

The probability of detecting a s-a-1 fault on signal line l is obtained by taking the product of the zero controllability and observability probabilities of l. Performing this operation provides

$$D_1(l) = C_0(l) O_0(l) \tag{8.16}$$

where $D_1(l)$ is the probability of detection for a s-a-1 fault on l. Likewise, the probability of detecting a s-a-0 fault on l is obtained by taking the product of the one controllability and observability probabilities associated with l. This product operation provides

$$D_0(l) = C_1(l) O_1(l) \tag{8.17}$$

where $D_0(l)$ is the probability of detection of a s-a-0 fault on l [34, 108, 109].

The observability and controllability probabilities are obtained by simulating the fault-free circuit for each input vector and gathering data concerning the signal activity of the DUT. The signal activity along with the structure of the DUT is used to calculate a probability estimate for the controllability and observability of each signal line [108, 109]. The detectability of stuck-at faults on each signal line is then determined by evaluating Equation (8.16) and Equation (8.17)

The coverage estimation begins by using the approximate fault detection probabilities given by Equation (8.16) and Equation (8.17) to calculate the probability of detecting a given fault over the entire input vector set I. The derivation of the single fault detection begins by calculating the probability of undetection of a fault for n input vectors. If the probability of detecting a single fault is given as x then the probability of undetection after n input vectors are applied is

$$U(n) = (1-x)^n \tag{8.18}$$

where $U(n)$ is the probability of undetection. Equation (8.18) is derived by assuming that the probability of detection is independent for each input vector. The independence assumption is valid for combinational circuits but is probably questionable for sequential circuits. The probability of detection for a given fault is given as

$$X(n) = 1 - U(n) = 1 - (1-x)^n \tag{8.19}$$

where $X(n)$ represents the probability of detecting a given fault over a given input vector set [109, 108]. The fault coverage estimation is obtained by summing the individual detection probabilities and dividing by the total number of faults contained in the fault list. Performing this summation and division operation provides

$$C = \frac{1}{m} \sum_{i=1}^{m} X_i(n) \tag{8.20}$$

where m is the number of faults in the fault list. Jain notes that the estimate provided by Equation (8.20) is biased in nature and presents a technique for eliminating the bias [109]. The unbiased coverage estimate equation is very similar in form to Equation (8.20).

The major limitation of probabilistic techniques is that the coverage estimate is approximate in nature and it is difficult to determine the accuracy of the estimate. The only technique available for gaging the accuracy of approximate coverage estimates based on probabilistic fault detection is to estimate the fault coverage via exact methods. Specifically, fault simulation must be performed to

estimate the fault coverage. Several example circuits are evaluated via probabilistic and exact methods in [109]. The results of this study indicate that probabilistic coverage estimates are within four percent of coverage estimates obtained via exact fault simulation. However, it is difficult to conclude that all DUTs which are evaluated using probabilistic fault detection will provide a coverage estimate that is accurate to ±4%. In fact, it is impossible to say how accurate probabilistic fault detection will be in general.

Another use for probabilistic fault detection is as a guide to select faults which have a high probability of being detected for evaluation via fault simulation. The purpose of combining approximate and exact fault simulation techniques is to minimize the amount of fault simulations for faults which have a low probability of being detected for the current input vector. This combined technique is computationally efficient since approximate fault evaluation has a significantly lower computational cost than exact fault simulation.

The combined approximate and exact fault evaluation process begins by simulating the fault-free DUT to generate the required signal activity data. The estimate on the controllability and observability probability is then calculated. The detection probability of each fault in the fault list is calculated next. All faults which have a detection probability greater than a threshold value such as 0.5 are grouped together for fault simulation. The primary advantage of the combined method is that the approximate fault detection mechanism has a low computational cost. The approximate information can then be used to eliminate from consideration the faults which have a low probability of being detected. A significant savings in fault simulation resources is obtained by eliminating the faults from the fault list which have a low probability of detection. This combined technique is used to evaluate an Ultra Large Scale Integrated (ULSI) circuit in [75]. Additionally, a hardware accelerator is employed to speed the fault simulation of the ULSI DUT.

9. Test Pattern Generation Overview

The generation of the test pattern set which is used to evaluate the DUT is referred to as Test Pattern Generation (TPG). There are numerous TPG techniques, however, the techniques can be grouped into three major categories: (1) Deterministic Automatic Test Pattern Generation (DATPG), (2) Random Automatic Test Pattern Generation (RATPG), and (3) Manual Test Pattern Generation (MTPG). A graphical depiction of the three major TPG categories and several subcategories is included as Figure 9.1. The fault grading process is impacted by the structure of the DUT, the fault model employed, and the size and quality of the test pattern set. The structure of the DUT is determined by the function that the DUT performs, the design for test strategy employed by the designer, and the technology employed to implement the DUT. The fault model employed to represent DUT faults is either selected by the designer or is specified by the end user of the DUT. The selection of the input vectors which test the DUT is a topic which has received a large amount of
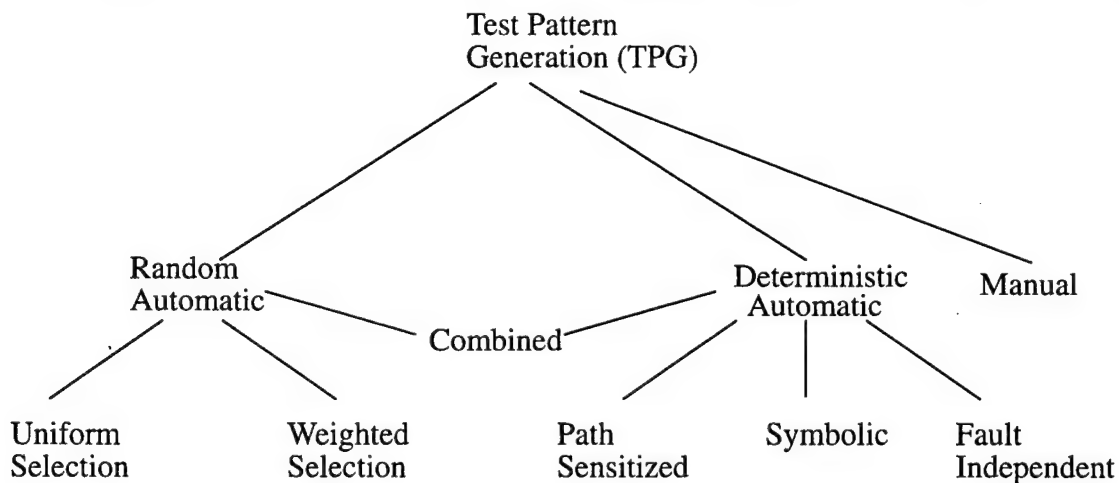
Figure 9.1. **Types of test pattern generation techniques**

attention. The survey of ATPG is included as Section 9.1 while the RATPG survey is presented in Section 9.2. The Combined TPG technique is described in Section 9.3. Likewise, the MTPG survey is presented in Section 9.4.

The TPG survey does not exhaustively cover the literature. Space and time constraints prevent an exhaustive overview of the TPG state-of-the-art. However, the TPG overview is complete in the sense that all major techniques of TPG are presented in this report. Numerous incremental improvements to major TPG techniques are not described in this report.

9.1. Automatic Test Pattern Generation

ATPG techniques determine a test pattern and provide a set of faults detected by the test pattern. The set of detected faults determined by ATPG typically is not complete. Specifically, fault simulation is required to determine the complete set of faults detected by an ATPG produced test pattern. ATPG methods can be delineated into three categories: (1) path sensitized methods, (2) symbolic, and (3) fault independent techniques. The following three subsections describe the various types of ATPG techniques.

9.1.1. Path Sensitization Methods

The primary purpose of an ATPG algorithm is to determine an input vector which detects a given fault in a DUT. In general, the path sensitized ATPG process can be envisioned as a three step algorithm: (1) activate the fault by setting the inputs of an internal component to the appropriate values, (2) justify the internal signal values to primary inputs, and (3) propagate the error caused by the activated fault to one or more primary outputs. The accomplishment of each one of these steps varies based on the implementation of a particular path sensitized ATPG algorithm. Concep-

tually, an ATPG algorithm is performing a search of the input space of the DUT to locate an input vector which detects a particular fault. Most, if not all, path sensitized ATPG methods are complete in the sense that they will prove through an exhaustive search that either no input vector exists that detects a given fault, or the technique will locate an input vector that detects the fault. The input space to be explored grows exponentially as the number of inputs to the DUT are increased. The time required to exhaustively search a large input space for a DUT can become excessive for even the most efficient ATPG algorithm. For this reason, most path sensitized ATPG algorithms allow the designer to specify the amount of input space evaluation which is allowed before the path sensitized ATPG algorithm declares a fault hard to detect and stops the input search process. A brief overview of the evolution of path sensitized ATPG algorithms is described next.

One of the first ATPG techniques was developed by Roth and is referred to as the D-algorithm [181]. The D-algorithm is based upon the notion of a 5 valued logic which is referred to by Roth as D calculus. The five values used by D calculus are 0, 1, D, $\bar{D}$, and x. The D and $\bar{D}$ symbols represent signal values which are erroneously 0 and 1, respectively. The D-algorithm first attempts to justify the inputs to the component that activates the fault. The current status of the justification process is represented in the D-algorithm by the J frontier. Once the inputs to the faulty component are justified then the effect of the fault is propagated to one or more primary outputs. The current status of the propagation is stored and is referred to as the D frontier. Once the entire J frontier reaches the primary inputs and the D frontier contains one primary output with either a D or $\bar{D}$ value then the D-algorithm has found a test pattern that detects the fault of interest.

The justify and propagate steps are sufficient for all DUTs that do not contain reconvergent fanout. If the DUT does not contain reconvergent fanout then each input of the DUT can be justified independent of all other inputs. However, when a DUT contains reconvergent fanout then the inputs to the DUT cannot be justified independently. When reconvergent fanout is present then the justification and propagation steps require a decision process. Whenever there are multiple alternatives to justifying a signal or propagating an error then the decision process selects an alternative to try. If the selected alternative is not successful then the decision process selects another untried alternative and the process is repeated. The selection of an alternative and the evaluation process is continued until either an alternative is found which provides the desired solution or all alternatives are considered. The retry mechanism is referred to as backtracking. The D-algorithm uses a form of backtracking to handle reconvergent fanout. Additionally, the D-algorithm is complete; that is, the D-algorithm will exhaustively search the input space of the DUT to determine that either no test pattern exists to detect the fault or locate a specific test pattern which detects the fault.

One of the problems associated with the D-algorithm is how the internal signal search space is explored. For example, consider the case where an error has k possible propagation paths. The D-algorithm will search all $2^k - 1$ different combinations of paths before it can determine that the

error cannot propagate to an output. The major drawback to the D-algorithm search process is the amount of computer resources required to evaluate all error propagation permutations to demonstrate that the error will not reach a primary output. Figure 9.2 is included to illustrate this concept. The example diagram depicts a D-frontier which contains k elements. The D-algorithm attempts to propagate each of the elements in the D-frontier individually. If the individual element propagation is unsuccessful then the D-algorithm attempts to propagate all combinations of two elements in the D-frontier. This search process is continued until one element of the D-frontier propagates to an output or until all possible $2^k - 1$ propagation permutations are evaluated.

The 9-V algorithm extends the D-algorithm by using nine signal values versus five to minimize the number of evaluations required to determine if error propagation is possible[45]. The nine signal values used by the 9-V algorithm are: 0, 1, D, $\overline{D}$, $0/D$, $0/\overline{D}$, $1/D$, $1/\overline{D}$, and x. The dual value signals such as $0/D$ represent that a signal can have either a correct value or an erroneous value. For the signal value $0/D$ the correct value is 0 while the erroneous value is D. The additional four signal values are used to assist the decision process to minimize the number of evaluations. For example, if an error has k possible propagation paths then the 9-V algorithm evaluates only k possible paths where the D-algorithm requires the evaluation of $2^k - 1$ permutations of error propagation paths. The additional signal information associated with the 9-V algorithm allows the search process to be performed in a much more efficient fashion than the D-algorithm. Conceptually, all
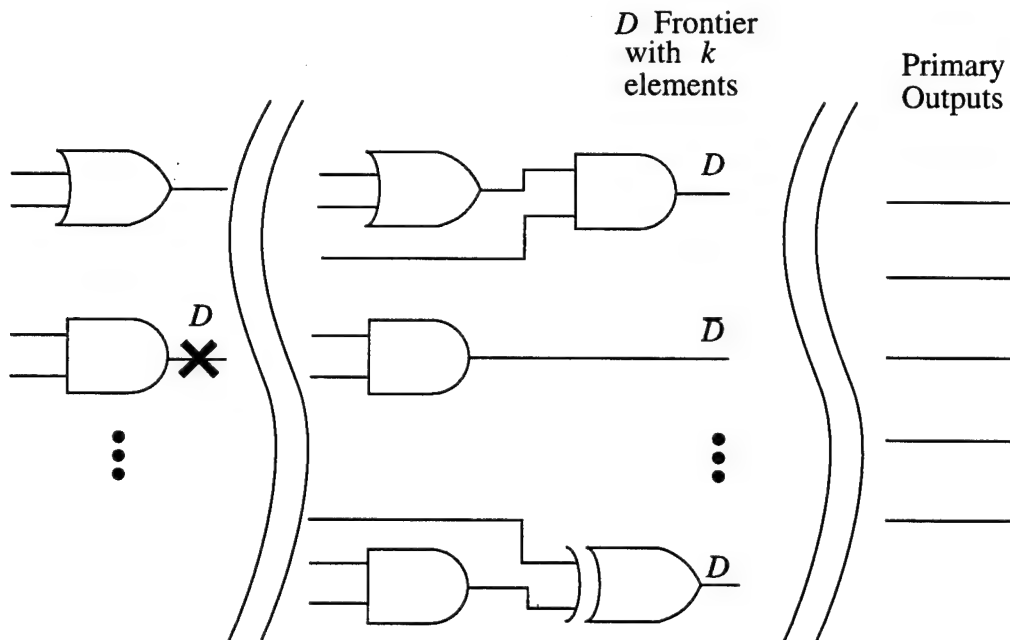

Figure 9.2.   D algorithm example demonstrating the D frontier.

possible permutations of the D-frontier for a single error propagation path depicted in Figure 9.2 are evaluated with a single pass. The additional signal information associated with the 9-V algorithm allows the certain permutations of the D-frontier which cannot propagate the error to an output to be eliminated from consideration.

Conceptually, the D-algorithm and 9-V search process explores all possible internal signal values in the attempt to determine an input vector which detects a given fault. The search process entails making decisions about internal signal values which then must be propagated and justified. A more intuitive search process entails making decisions about possible DUT input values. If the decision process is moved to the circuit inputs then much of the overhead associated with making decisions concerning internal signal values can be eliminated. The Path Oriented Decision Making (PODEM) algorithm performs decisions on the primary inputs. The propagation of the D frontier in PODEM is performed in a similar fashion as the D-algorithm. The justification process for PODEM uses a direct search based on sensitizing paths from the primary input to the internal signal which requires justification. The path sensitization process is performed in a recursive fashion starting with the internal signal which must be justified. The gate which sets the signal to be justified is then evaluated to determine a possible input value required to produce the desired justified signal value. The selected input to the gate is then evaluated in a recursive fashion until a primary input is reached. The first primary input value is set to the appropriate value and the DUT is then simulated to determine if the test generation process is complete. If the test generation is not complete then the propagation of the D frontier is performed next. A signal is selected for justification to propagate the D frontier and another recursive backtrace operation is performed to set another primary input value. The DUT is then simulated with the two primary input values. The process of backtracing to set a primary input and simulating the DUT with the set of defined primary inputs is repeated until the test generation process is completed. The test generation process is complete when the D frontier contains a primary output signal which has either a D or $\bar{D}$ value [79, 83].

PODEM has several advantages over the D-algorithm. Specifically, PODEM does not need: (1) a consistency check since conflicts can never occur, (2) a J frontier since there are no values which require justification, and (3) backward implication because values are only propagated forward [1]. Another key advantage of PODEM is the implicit state restoration process. In the D-algorithm when a decision is made and is found to be incorrect then the state of the DUT must be restored to the value which existed before the decision was made. For large circuits the state restoration generates a large amount of overhead. With PODEM, when an incorrect decision is made the incorrect input is removed and a new input value is applied. The DUT is then simulated. Thus, PODEM requires no explicit state restoration because the internal signal values of the DUT are determined by the simulation. Additionally, the decision space explored by PODEM is stored in the form of a decision tree. The decision tree is used as a bookkeeping measure to keep track of the input space

that has been explored by PODEM. Thus, the decision tree allows all input vectors which could detect the fault to be explored.

The Fanout-Oriented TPG (FAN) algorithm [69] extends the PODEM algorithm with two improvements: (1) backtracing in FAN may stop at internal lines, and (2) a multiple backtracing strategy is used to attempt to satisfy a set of signal objectives simultaneously [1]. The internal lines where FAN stops the backtracing process are the outputs of a FFR and are referred to as head lines. A final postprocessing step is employed with FAN to determine the values on the primary inputs of the FFR based on the values of the head lines through a reverse implication process. The multiple backtracing process speeds the input search process by minimizing, in general, the number of backtracing passes.

The Structure-Oriented Cost-Reducing Automatic TESt pattern generation system (SOCRATES) improves upon the FAN algorithm in several ways [188]. The first enhancement deals with increasing the effectiveness of the FAN multiple backtracing method. Heuristics derived from the structure of the DUT are used to improve the efficiency of the multiple backtracing process by eliminating needless backtracing attempts. The second improvement is achieved through using an implication procedure to determine the logic value of as many signals as possible. The third improvement involves using a unique sensitization procedure when the D frontier contains a single signal.

9.1.2. Fault Independent Methods

The key feature of fault independent methods is that the algorithm is not attempting to detect a specific fault but to determine a large fault set detected by the generated test pattern. Most fault independent methods require a gate-level model and begin by selecting an output value for the DUT. Each output value of the DUT is selected to satisfy a path sensitization criteria. Specifically, the gate which sets the output value must have a controlling input. An input is considered to be controlling if changing the input causes the output of the component to change value. For example, a two input AND gate with a 10 input pattern has the 0 input as a controlling input. The fault independent methods search for input combinations on gates which cause long sensitized paths. The sensitized path is determined by working backwards through the DUT and noting the continuous chain of controlling inputs. Changing the value of any of the controlling inputs on the continuous chain causes a DUT output error [13, 28, 209, 230].

The fault independent methods share several characteristics with Circuit Structure Based (CSB) techniques. Specifically, the concept of critical signals which is used with the CPT technique is employed with most fault independent TPG techniques. The general algorithm associated with fault independent test pattern generation is a two step process [1]:

       1. Select a primary output and assign it a critical 1 or 0 value

2. Recursively justify the primary output by justifying any critical gate outputs with critical values on the gate inputs

The recursive justification process is aided via the use of tables referred to as critical cubes. A critical cube table lists all of the critical input combinations associated with a given component type. The critical cube associated with a three- input AND gate [1] is included as Table 9.1 to facilitate discussion. The three inputs to the AND gate are given as A, B, and C and are represented by the first three columns of Table 9.1. Likewise, the output of the AND gate is given as O and is represented by the fourth column of Table 9.1. The critical values associated with a given input pattern which is represented by a row in Table 9.1 are denoted by bold face type. For example, the critical input value associated with a 011 input pattern is the zero input.

Table 9.1. Three input AND gate critical cubes

A	B	C	O
1	**1**	**1**	**1**
0	1	1	**0**
1	**0**	1	**0**
1	1	**0**	**0**

During the recursive evaluation process of a given component in the DUT the critical cube table is reference numerous times. Specifically, each row in the critical cube table for a given component which contains the necessary critical output value is evaluated to determine a set of test patterns which have one or more critical paths which span from one primary output to at least one primary input. For example, a three-input AND gate which requires a zero critical output is evaluated for three critical input patterns; that is, the second to forth rows of Table 9.1 are used.

The workings of the generic fault independent TPG technique are best illustrated by example. A simple four-input circuit shown in Figure 9.3a is used to illustrate the process. The example TPG process begins by selecting the output G to have a 0 critical output value. The critical cube table associated with a two input AND gate is referenced, and the first row which contains a zero output is selected to specify the critical input value. After performing this table look up operation E is set to 1 which is a noncritical input while F is set to 0 which is a critical input. Another critical cube look up operation is performed to determine the critical input on the two input XNOR gate which satisfies $F = 0$. The first entry in the two input XNOR critical cube table which has a zero output is selected which results in C being set to a 0 noncritical value while D is set to a 1 critical value. The A and B gate input values are selected next to assure the value of $E = 1$. A reverse implication procedure is used to determine that $A = 0$ and that B can have any value; that is, $B = x$. The reverse implication procedure is performed by using the inverse functional mapping of a compo-

- 85 -
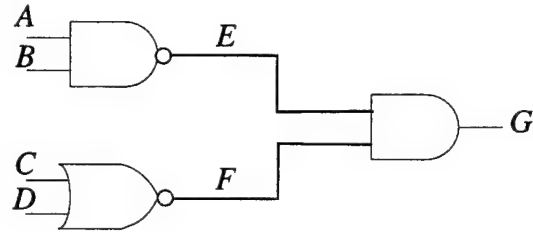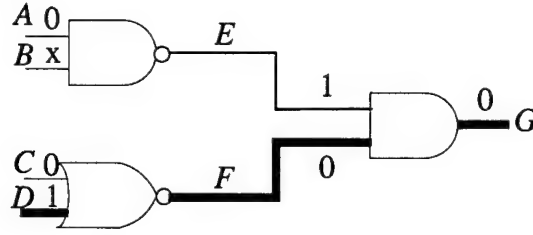
Figure 9.3a

Figure 9.3b
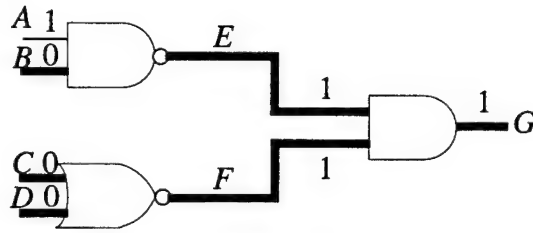
Figure 9.3c

Figure 9.3.   Example of fault independent test pattern generation.

nent to determine the input patterns which satisfy a given output condition. The critical path specified by the example test pattern is denoted by a bold line in Figure 9.3b. Any fault which is on the critical path and produces an error is detected. Another example is included as Figure 9.3c to illustrate the independent fault TPG when the critical output value is set to one; that is, $G = 1$.

For a DUT which does not contain reconvergent fanout the aforementioned algorithm is sufficient. Several additional issues must be addressed for a general DUT which contains reconvergent fanout. Specifically, reconvergent fanout introduces three additional problems which must be addressed: (1) signal value conflicts, (2) self-masking of fault effects, and (3) multiple path sensitization. The solution to these issues requires that an algorithm be developed which searches the internal signal space of the DUT. The algorithm exploits numerous features of the justification process used by path sensitive ATPG algorithms. For this reason, a discussion of the internal search algorithms employed by fault independent TPG is not presented in this report.

9.1.3. Symbolic Techniques

The symbolic ATPG category determines test patterns for gate-level designs by manipulating equations. A symbolic technique begins by converting a gate-level model to a set of Boolean expressions. The Boolean expressions are then manipulated using a predefined sequence of operations to produce a final set of equations. Each resulting equation represents an input condition which when satisfied detects a given fault. Specifically, an input which causes a resulting equation to be logical one produces a test pattern which detects a given fault.

Typically symbolic TPG is performed using Boolean difference methods. The Boolean difference of a Boolean function can be used to determine the test pattern necessary to test for faults associated with an input to the Boolean function. The Boolean difference for a general Boolean function is defined as

$$\frac{d}{dx_i}f(x_1, x_2, ..., x_n) = f(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) \oplus f(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n) \quad (9.1)$$

where $\oplus$ represents an XOR operation [119, 140]. To illustrate the Boolean difference concept consider [140]

$$f = yz + \bar{y}(w + x) \quad (9.2)$$

To detect a fault on w requires the generation of $f_{\bar{w}}$ and f_w, where $f_{\bar{w}}$ and f_w are Equation (9.2) with $w = 0$ and $w = 1$, respectively. Performing this evaluation yields

$$f_{\bar{w}} = yz + x\bar{y}$$
$$f_w = \bar{y} + z \quad (9.3)$$

The Boolean difference to detect faults with w in the function represented by Equation (9.2) is obtained by XORing $f_{\bar{w}}$ and f_w. Performing this operation provides

$$\frac{df}{dw} = \bar{x}\bar{y} \quad (9.4)$$

Thus, a faulty w is detected by setting x and y to 0.

The test vector for stuck-at faults is easily derived using the Boolean difference concept [119, 140]. Specifically, a test pattern which detects a stuck-at zero fault for x_i is generated by solving

$$x_i\frac{d}{dx_i}f(x_1, x_2, ..., x_n) = 1 \quad (9.5)$$

Likewise, a test pattern which detects a stuck-at one fault for x_i must satisfy

$$\bar{x}_i\frac{d}{dx_i}f(x_1, x_2, ..., x_n) = 1 \quad (9.6)$$

Using the aforementioned example for the Boolean difference of w which is given by Equation (9.6) the stuck-at zero test pattern is given as

$$w\frac{df}{dw} = w\bar{x}\bar{y} = 1 \qquad (9.7)$$

Likewise the stuck-at one test pattern for w is given as

$$\bar{w}\frac{df}{dw} = \bar{w}\bar{x}\bar{y} = 1 \qquad (9.8)$$

A major problem with using the Boolean difference technique is that it is an equation based technique that requires a large amount of symbolic manipulation. The symbolic manipulations are difficult to implement in an efficient algorithm that can readily handle large circuits. One method for overcoming this limitation is to compute $f_{\bar{x}_i}$ and f_{x_i}. The XOR operation to complete the Boolean difference is not performed. Instead, the two partial equations $f_{\bar{x}_i}$ and f_{x_i} are used to assist the input search process. The ATPG algorithm presented in [124] uses this approach. The CATAPULT ATPG algorithm, conversely, uses Boolean difference information to measure the observability of signal locations in the DUT. The observability information is then used in conjunction with controllability information to determine the input test pattern. The CATPULT algorithm is designed for use with hard to detect faults [71].

This section provides a brief overview of ATPG techniques. ATPG is currently and has been an active area of research for many years. The overview covers the salient points of ATPG but does not provide a detail about every ATPG algorithm described in literature. For a more detailed overview of ATPG the reader is encouraged to read Chapter 6 of [1].

9.2. Random Automatic Test Pattern Generation

The RATPG process begins by selecting an input pattern at random. The selected input pattern is evaluated using fault simulation to determine the faults which are detected by the input pattern. The detected faults are removed from the fault list. The next input pattern is selected at random, and the fault simulation and fault dropping process is repeated. The addition of vectors to the test pattern set is completed when the number of undetected faults drops below a certain threshold [35, 41, 124, 192].

For RATPG to be feasible, a fast computationally efficient fault simulator is required. However, RATPG has one fundamental problem which is, the decreasing number of detected faults for each additional test pattern. At the beginning of the RATPG process, each test pattern detects a large number of faults. As additional test patterns are evaluated, the number of detected faults begins to decrease. The decrease in the number of faults detected by each additional test pattern is attributed to two factors: (1) the size of the undetected fault set is smaller with each additional test pattern, and (2) the remaining undetected faults are random pattern resistant. A fault is random pattern resistant if there exists only a small number of input vectors which detect the fault. In the general case, a DUT with n inputs has 2^n possible input vectors. For the sake of discussion assume that a

given fault is detected by m input vectors. If we assume that the input vectors are selected using a uniform distribution then the probability of selecting one of the m input patterns which detect the fault is

$$p = \frac{m}{2^n} \qquad (9.9)$$

where p is the probability of selecting an input pattern which detects the fault. Equation (9.9) applies to the case where sampling with replacement is employed. Sampling without replacement has a selection probability that decreases as the number of test patterns in the test set increase [63].

A random pattern resistant fault is defined mathematically as

$$\frac{m}{2^n} \ll 10^{-4} \qquad (9.10)$$

Specifically, the probability of selecting an input pattern which detects a random pattern resistant fault is very close to zero. The worst case scenario is a random pattern resistant fault which is detected by only one input vector; that is, $m = 1$.

A graphical representation of a hypothetical random pattern resistant fault scenario is shown in Figure 9.4. The example figure indicates the relationship of m for each fault f_i. Specifically, the y axis of Figure 9.4 represents m while the x axis represents f_i. There are three random pattern resistant faults and one redundant fault depicted in Figure 9.4.

9.2.1. Uniform Input Selection

The random selection of input vectors requires some type of random number generator. In software, an acceptable random number generator is provided by the math libraries associated with
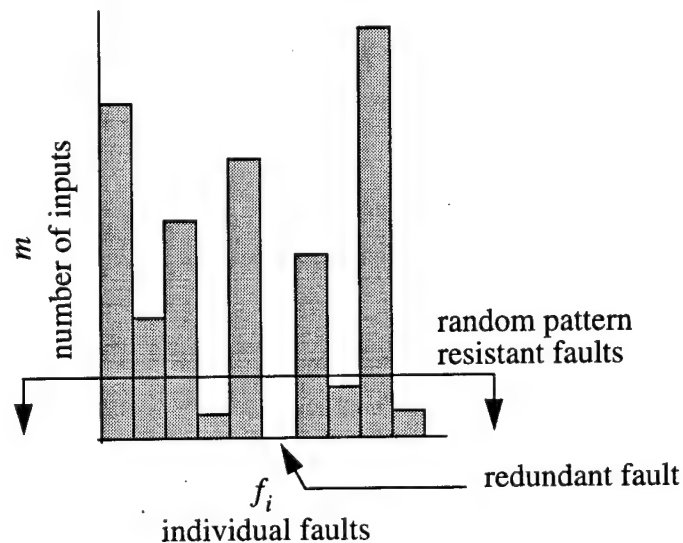

**Figure 9.4. Histogram of number of inputs which detect faults.**

high-level languages. For example, the *rand()* function in the C programming language can be used to generate the random patterns. In hardware, a Linear Feedback Shift Register (LFSR) is typically employed to generate the random vectors [140]. Typically, a uniform distribution is used to generate the random test patterns; that is, all test patterns are equally likely. One problem with the uniform distribution is that the resulting test pattern set can become quite large.

A graphical depiction of the probability mass function (*pmf*) associated with selecting an input pattern which detects a general fault f_i is included as Figure 9.4. Specifically, the *pmf* for selecting a test pattern which detects f_i is calculated by dividing m_i by n when uniform input sampling is employed. A large number of randomly sampled inputs are required to detect random pattern faults because the probability of selecting a test pattern which detects the random pattern resistant fault is very small.

9.2.2. Weighted Input Selection

Analyzing the structure of the DUT to determine a better input distribution is one way to overcome the random pattern resistant problem. Three different techniques are used to select the input weights [21, 62, 228]. The details associated with the three techniques are introduced after a general description of weighted input selection is provided. The basic concept is to adjust the input distribution by weighting each input bit based on the structure of the DUT. The approach is to change the sampling distribution so that the probability of selecting an input pattern which detects a random pattern resistant fault is increased. Figure 9.5 represents the *pmf* which results from applying weighted input selection for selecting an input pattern that detects a given fault f_i. Specifically, $p(s_i)$ represents the probability of selecting an input pattern which detects f_i. The end result of applying weighted input pattern selection is the flattening of uniform selection *pmf*. An example of the *pmf* flattening can be observed by comparing the uniform *pmf* depicted in Figure 9.4 to the weighted *pmf* shown as Figure 9.5. Also, the probability associated with detecting a redundant fault is not increased with weighted selection because there exists no input pattern which will detect redundant fault. Thus, no amount of input weighting will cause the probability of detecting a redundant fault to increase from zero.

The three weighted input selection techniques are very similar in the manner in which the input weights are calculated. The first algorithm presented is based on one fundamental premise; that is, the probability of detecting a faulty DUT is maximized when the output entropy of the DUT is maximized [10]. The term premise is used because the relationship between maximum output entropy and maximizing the probability of detecting a fault has never been proven. This technique is referred to as the Maximum Output Entropy (MOE) method. The output entropy of a DUT is maximized if the inputs are selected such that the probability of a DUT output being either a 1 or 0 is 0.5. The problem is reduced to calculating the input *pmf* which maximize the output DUT
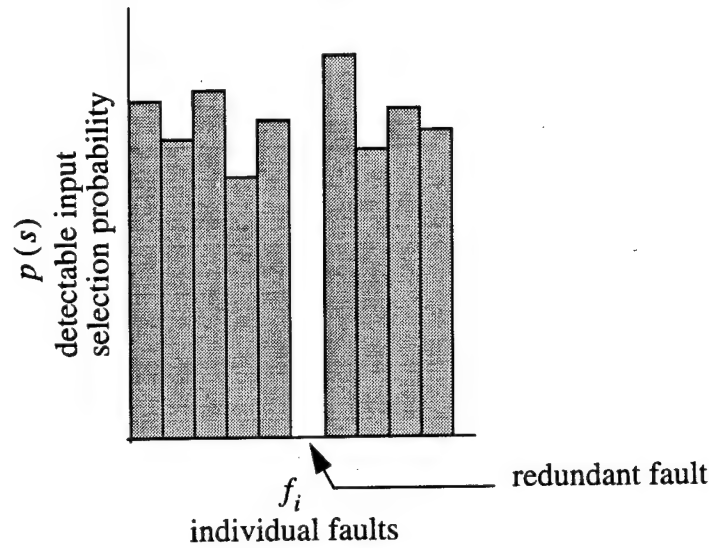
Figure 9.5. **Example of weighted input selection probability mass function.**

entropy. A two input AND gate is considered as an example to illustrate this concept. If all inputs to a two input AND gate are equally likely then the probability of producing a 1 output is 0.25; that is, only one input out of four possible inputs causes a one to be produced. The entropy of the output is maximized if the probability of producing a 1 output is 0.5. If the two inputs to the AND gate are assumed to be independent then the entropy is maximized when

$$P(I_1 = 1)P(I_2 = 1) = 0.5 \tag{9.11}$$

where $P(I_1 = 1)$ is the probability that the I_1 input is a logic one, and $P(I_2 = 1)$ is the probability that the I_2 input is a logic one. The simplest scenario to consider is when

$$P(I_1 = 1) = P(I_2 = 1) \tag{9.12}$$

Using Equation (9.12) to simplify Equation (9.11) provides

$$P(I = 1) = \frac{1}{\sqrt{2}} \tag{9.13}$$

The generic problem of determining the input *pmf* for a single output fanout free DUT is now considered. A simple three gate example is used to illustrate this concept and is included as Figure 9.6. The approach is to start at the gate which produces the primary output and calculate the input probabilities necessary to maximize output entropy. For the example circuit shown in Figure 9.6 the output gate is the two input AND gate labeled as 1. The input probabilities for gate 1 are specified by Equation (9.13).

The input probabilities for the gates which set the inputs to gate 1 are calculated next. For the sake of discussion the input probabilities for gate 2 are derived. Gate 2 is a two input OR gate. The

$$P(I=1) = 1 - \sqrt{\frac{\sqrt{2}-1}{\sqrt{2}}}$$

$$P(I=1) = \frac{1}{\sqrt{2}}$$

$$\boxed{2}$$

$$P(I=1) = 1 - \sqrt{\frac{\sqrt{2}-1}{\sqrt{2}}}$$

$$P(I=1) = \frac{1}{2}$$

$$\boxed{1}$$

$$P(I=1) = \sqrt{1 - \frac{1}{\sqrt{2}}}$$

$$\boxed{3}$$

$$P(I=1) = \frac{1}{\sqrt{2}}$$

$$P(I=1) = \sqrt{1 - \frac{1}{\sqrt{2}}}$$
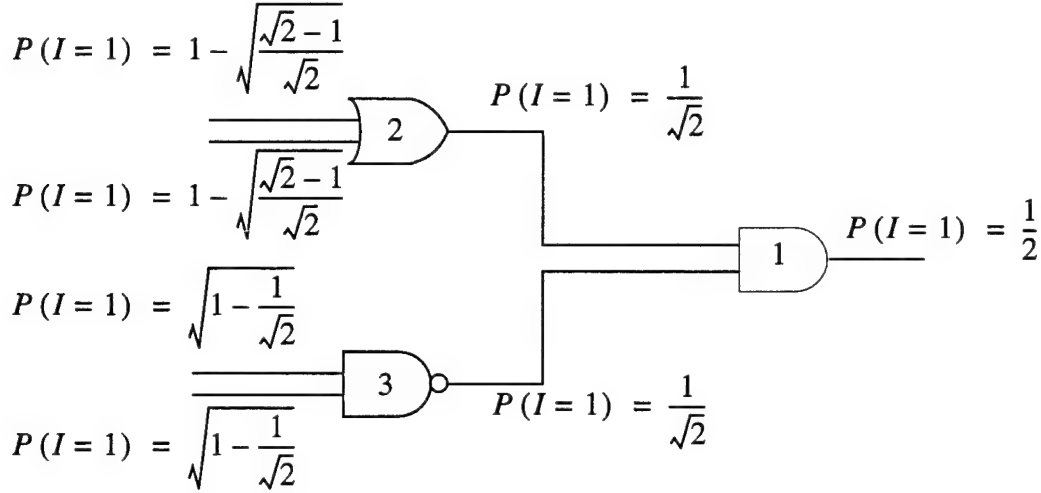
Figure 9.6.   Test pattern generation example for a hypothetical XOR tree.

output of a two input OR gate is zero if and only if both inputs are logic zero. The input to output probability relationship for a two input OR gate is

$$P(Out=0) = 1 - P(Out=1) = (P(I=0))^2 = (1 - P(I=1))^2 \tag{9.14}$$

The output probability for gate 2 is

$$P(Out=1) = \frac{1}{\sqrt{2}} \tag{9.15}$$

Equation (9.15) is used to eliminate one unknown from Equation (9.14) and the resulting equation is solved which provides

$$P(I=1) = 1 - \sqrt{\frac{\sqrt{2}-1}{\sqrt{2}}} \tag{9.16}$$

The input probabilities for gate 3 are derived by noting that a two input NAND gate produces a logical one output if and only if both inputs are logical zero. The input to output probability relationship for gate 3 is given as

$$P(Out=1) = (P(I=0))^2 = (1 - P(I=1))^2 \tag{9.17}$$

The output probability for gate 3 is

$$P(Out=1) = \frac{1}{\sqrt{2}} \tag{9.18}$$

Using the result provided by Equation (9.18) to eliminate a variable from Equation (9.17) and solving for $P(I=1)$ yields

$$P(I=1) = \sqrt{1 - \frac{1}{\sqrt{2}}} \tag{9.19}$$

The previous example demonstrates how input probabilities which maximize the output entropy of a single output fanout free circuit are calculated. Maximizing DUT output entropy by applying random inputs is based on the premise that the probability of detecting a fault in the DUT is also maximized if the output entropy is maximized. No mathematical proof exists which substantiates the maximum output entropy/maximum detection probability relationship. Specifically, Debany presents experimental results which indicate that the maximizing output entropy does not cause the probability of detecting a faulty DUT to be maximized [62]. Thus, the fundamental premise used to construct the MOE technique is questionable.

The general DUT can contain: multiple outputs, signal fanout, and reconvergent fanout. The single output fanout free technique must be extended to consider the general case. Unfortunately, properly handling signal fanout to calculate component input probabilities is a difficult problem. Each fanout signal will more than likely require a different probability to maximize output entropy. Unfortunately, there is no known closed form solution on how different fanout probabilities should be merged. The MOE technique does not address how to select the input weights for the general case. The technique presented in [10] provides the mathematical argument for using maximum entropy. Several example circuits are included and results are shown but no generic algorithm is presented by Agrawal. Debany in [62] notes the difficulty of using the MOE approach for nontrivial DUTs.

The second weighted input selection technique developed by Bardell, McAnney, and Savir uses approximate techniques to handle the fanout problem [21]. This technique is referred to as the weighted input selection 1 (WIS1) algorithm in this document. Instead of trying to maximize the output entropy the WIS1 technique uses a relationship which is proven mathematically as the starting point for the algorithm. Specifically, for AND and NAND gates the optimal input probability for infinitely long input streams is given as

$$P(I = 1) = \frac{(n-1)}{n} \tag{9.20}$$

where n is the number of inputs to the gate [21]. Likewise, for NOR and OR gates the optimal input probability for infinitely long input streams is

$$P(I = 1) = \frac{1}{n} \tag{9.21}$$

The WIS1 algorithm is very similar to the MOE. Specifically, a reverse traversal from the output of the DUT to the inputs of the DUT is performed in a depth first evaluation. The algorithm consists of an initialization phase and three step evaluation process. The initialization phase consists of assigning the input probabilities given by Equation (9.20) and Equation (9.21) to the inputs of the components which drive the primary outputs. The backward traversal process begins by cal-

culating the input probabilities for the components which drive the signals where the probabilities are known. The input probabilities for a k-input AND gate is

$$P(I = 1) = (P(Out = 1))^{1/k} \qquad (9.22)$$

Likewise, for a k-input OR gate the input probabilities is given as

$$P(I = 1) = 1 - (1 - P(Out = 1))^{1/k} \qquad (9.23)$$

For an INVERTER the input probability is

$$P(I = 1) = 1 - P(Out = 1) \qquad (9.24)$$

For a k-input NAND gate the input probability is

$$P(I = 1) = (1 - P(Out = 1))^{1/k} \qquad (9.25)$$

For a k-input NOR gate the input probability is

$$P(I = 1) = 1 - (P(Out = 1))^{1/k} \qquad (9.26)$$

For a fan-out with branch signal probabilities $P(Out = 1)_i \forall i \in \{1, 2, ..., k\}$ the stem signal probability is

$$P(I = 1) = \frac{1}{k} \sum_{i=1}^{k} P(Out = 1)_i \qquad (9.27)$$

The reverse traversal of the DUT continues until all components in the DUT are evaluated and the probabilities associated with the primary inputs are known.

The second step of the algorithm consists of recording the calculated primary input signal probabilities. The calculated primary input probabilities may or may not sum to one if the DUT contains reconvergent fanout. The third step of the algorithm calculates the weights needed to select the random input vectors to be applied to the DUT. The input weight for the jth primary input is given as

$$W(I = 1)_j = \frac{P(I = 1)_j}{\sum_{i=1}^{k} P(I = 1)_i} \qquad (9.28)$$

where $W(I = 1)_j$ is the input weight for the jth input [21].

The WIS1 technique handles the fanout problem by averaging the fanout branch probabilities to calculate the fanout stem probabilities as given by Equation (9.27). Likewise, the starting point of the WIS1 is based on two relationships which are proven to be correct mathematically. Thus, the WIS1 technique has a firmer theoretical foundation than the MOE technique and the WIS1 technique can be used for nontrivial circuits which contain fanout.

The third weighted input technique developed by Waicukauski [228] is similar to WIS1. The Waicukauski algorithm is referred to as Weighted Input Selection 2 (WIS2). One primary difference between the WIS1 and WIS2 techniques is that WIS2 employs integer based calculations to

calculated gate input probabilities. Both the WIS1 and MOE techniques use k^{th} root evaluation to calculate input probabilities. The manner in which fanout stem probabilities are handled also differs between the two techniques. Specifically, the WIS1 algorithm uses an averaging function (Equation (9.27)) to calculated the fanout stem while the WIS2 algorithm uses the largest fanout branch probability as the fanout stem probability [228]. The selection of the largest probability provides an approximate solution to determining input probabilities which maximize output entropy [228]. An overview of the low-level details of the approximate algorithm are omitted in this report due to space constraints.

A comparison of the three weighted input techniques is performed by Debany to determine the best weighted input selection technique in [62]. The comparison is performed using experimental techniques. Specifically, the experiment consists of evaluating a number of circuits to determine two factors: (1) if the three techniques provide equivalent results, and (2) which technique provides the best performance. The equivalent performance factor is determined using statistical based hypothesis testing. Likewise, the rank ordering is performed based on the number of randomly selected test vectors required to achieve a given test coverage. The technique which generated the fewest test vectors is considered best. For the circuits evaluated the MOE technique never generated the best results. For all cases where the coverage is 98% or greater either WIS1 or WIS2 technique provided the best performance [62].

9.3. Combined Test Pattern Generation

One method for overcoming the problem associated with random pattern resistant faults is to combine the RATPG and DATPG techniques [4, 35, 124]. RATPG is used until the number of faults detected for a new input pattern drops below a predefined threshold. DATPG is then employed to detect the remaining faults. The benefit of this combined technique is that RATPG is very computationally efficient for the initial test patterns. As the fault set becomes smaller and the number of detected faults per additional input vector becomes small the computational cost of RATPG increases dramatically. When the computational cost starts to become excessive with RATPG then DATPG is employed to detect the remaining random pattern resistant faults. The rate at which faults are detected from the undetected fault set slows as test patterns are added to the test pattern set with RATPG because the initial test patterns locate the easy to detect faults. Eventually the undetected fault set comprises only of random pattern resistant faults. The idea with combined TPG is to have an ATPG technique determine test patterns for the remaining set of random pattern resistant faults. Thus, RATPG is used to generate test patterns to detect the set of easy to detect faults. RATPG is far more efficient than DATPG techniques at detecting nonrandom pattern resistant faults (easy to detect faults). Using RATPG to generate test patterns for the set of easy to detect faults decreases the computational resources required for TPG. Conversely, RATPG tends to

become inefficient when used to detect random pattern resistant faults. DATPG techniques can locate a test pattern to detect a random pattern resistant fault more efficiently than RATPG. For these reasons, the combined technique exploits the best attributes of both RATPG and DATPG.

9.4. Manual Test Pattern Generation

The distinguishing characteristic of the MTPG technique is that the test patterns are selected manually by the designer. Typically, the first test patterns selected are also used by the designer to verify that the DUT has the correct functional mapping. Like RATPG, each new test pattern is evaluated via fault simulation to determine if any faults from the undetected fault set are detected. If no faults are detected by the new test pattern then the test pattern is discarded; otherwise, the new test pattern is added to the test pattern set. Fault grading is then employed to determine if the specified level of fault coverage is achieved. If the test pattern set achieves or exceeds the fault coverage specification then the MTPG process is ended, else the MTPG process is continued. Subsequent test patterns are selected after the designer performs some analysis of the DUT. After each test pattern is selected a fault simulation is performed to determine if any undetected faults are detected. If no additional faults are detected then the new test pattern is discarded, else the new test pattern is added to the set of test patterns. The manual selection process continues until the number of undetected faults reaches an acceptable level.

10. Applicability of Existing Techniques for VHDL Fault Simulation

This section provides an analysis of existing fault simulation techniques described in Sections 5 and 6 to determine their applicability to VHDL-based fault simulation. Before the analysis is performed the concept of what is meant by VHDL fault simulation is defined. After this concept is fully defined then the analysis is performed.

There are several attributes associated with VHDL-based fault simulation. The first of these attributes is the ability to perform fault simulation on a VHDL model with any compliant VHDL simulator. The simulator independent property implies that the fault insertion methodology must use features associated with the VHDL language and not VHDL simulator specific features. The second attribute deals with adding fault insertion to existing VHDL models. The fault insertion technique must be performed in such a fashion where there are a minimum or zero changes required to the VHDL model. The VHDL fault insertion technique developed by DeLong satisfies this model modification attribute [64]. The third attribute of VHDL-based fault simulation is that the fault insertion must be performed in a consistent fashion across all levels of modeling abstraction. The method presented by DeLong in [64] satisfies the fault insertion consistency attribute. The aforementioned three attributes define what is meant by the term VHDL fault simulation.

The analysis of the attributes associated with existing fault simulation techniques described in Sections 5 and 6 is performed to determine which fault simulation methods can be employed for VHDL-based fault simulation. The attributes of fault simulation methods are divided into two categories. The first category is whether the fault simulation technique supports multiple levels of design abstraction. Specifically, the fault simulation technique must be able to insert faults at all levels of design abstraction to satisfy this attribute category. The second category is related to the type of simulation performed; that is, value corruption via fault insertion or symbolic fault simulation. A common attribute of symbolic fault simulation is that faults are stored in a data structures. The stored faults are then manipulated when a component is evaluated to determine which faults produce a component output error. For example, concurrent fault simulation requires a custom data structure to represent each component in the DUT and dynamic memory allocation to store the fault list associated with each component.

An overview of whether a fault simulation technique satisfies each attribute category is included as Table 10.1. The columns of Table 10.1 represent each attribute category while the rows are associated with a specific fault simulation technique. A ✓ symbol in a table cell denotes that a given fault simulation method has a given attribute. For example, the first row of Table 10.1 represents serial fault simulation which has the attribute of supporting multiple levels of design abstraction denoted by a ✓ symbol in the second column. Additionally, Table 10.1 represents a summary of requirements for the manner in which the existing set of fault simulators described in literature have been implemented. In theory, future developments in fault simulation could change the results presented in Table 10.1. Thus, Table 10.1 represents the attributes associated with the current state-of-the-art for fault simulation.

PPSFP/SPMFP fault simulation is typically used to evaluate gate-level models. Models with higher levels of abstraction such as algorithmic-level models or behavioral microprocessor models are typically not evaluated using parallel fault simulation. High-level models do not map efficiently to the parallel fault simulation paradigm. Specifically, for parallel fault simulation to be practical the W parallel DUTs must be simulated using W-bit computer words where each bit represents a different DUT. The parallelization process for gate-level models is relatively straight forward. The DUT is transformed into a model comprised of one input NOT gates and two input AND, OR, NOR, NAND and XOR gates. The modified model is then translated to machine instructions that execute on a host processor. Each machine instruction has W-bit input and output operands. The level of effort required to implement a VHDL-based fault simulation tool using a parallel fault simulation technique for gate-level models is small. However, the level of effort required to implement VHDL-based parallel fault simulation for models with higher levels of abstraction is quite high. The primary difficulty is determining an efficient mechanism for translating behavioral VHDL models such as a microprocessor models into W parallel behavioral models. Specifically, signifi-

Table 10.1. Features and requirements of existing fault simulation techniques

Features and Req./ Simulation Technique	Multiple Levels of Design Abstraction	Symbolic Fault Simulation
Serial	✓	
Parallel Pattern		
Parallel Fault		
Deductive		✓
Concurrent	✓	✓
Differential		
Hierarchical Concurrent	✓	✓
Hierarchical Serial	✓	

cant amounts of research and development would be required to implement VHDL-based parallel fault simulation which can be used at all levels of design abstraction. Also, the behavior of an event driven SPMFP fault simulation requires a significant amount of book keeping for the simulation to be performed correctly. For this reason the overhead associated with event driven SPMFP fault simulation is quite large. For this reason PPSFP and SPMFP fault simulation is not a preferred technique for VHDL-based fault simulation.

Deductive and concurrent fault simulation techniques suffer a similar problem. Both concurrent and deductive fault simulation methods propagate fault lists when the internal signals in the DUT are updated. The size of the fault list associated with each signal varies based on the input applied to the DUT and the location of the signal in the DUT. For this reason the fault list associated with each signal is typically stored in a data structure where the memory is dynamically allocated during fault simulation. Specifically, the dynamic memory allocation/deallocation occurs when a signal value is updated during the fault simulation of the DUT. The individual faults in the fault list are typically stored in data structures. The fault list propagation attributes associated with the concurrent and deductive simulation techniques require special simulator features; that is, all existing concurrent and deductive simulators are implemented using a custom simulator. A large amount of

research and development effort would be required to determine a VHDL implementation of either the concurrent or deductive fault simulation techniques. For this reason both concurrent and deductive techniques are not a preferred method for VHDL-based fault simulation.

Differential fault simulation also has some fundamental limitations. Specifically, differential fault simulation is primarily used for sequential circuits modeled at the gate-level. The simulation technique is based on evaluating the combinational portion of the DUT for each undetected fault. Each simulation requires that the state of the sequential circuit be restored before the next fault is evaluated. A significant amount of research and development is required for VHDL-based differential fault simulation to determine: (1) a technique to perform state restoration, and (2) a method to extend the differential technique for use with high-level behavioral models. The difficulty associated with solving the aforementioned two issues makes differential fault simulation an undesirable method for VHDL-based fault simulation.

Existing hierarchical fault simulators use one of two different fault simulation techniques to inject faults: (1) concurrent fault simulation, and (2) serial fault simulation. Concurrent based hierarchical fault simulation has the same problems associated with concurrent fault simulation. Specifically, the fault list propagation requires the dynamic allocation/deallocation of memory for the fault list data structures. Hierarchical concurrent fault simulation is difficult to implement as a VHDL-based technique because of the development effort associated with the fault list propagation attribute. For this reason, hierarchical concurrent fault simulation is not a preferred technique for VHDL-based fault simulation. Thus, serial based hierarchical fault simulation is the only existing fault simulation technique which requires a reasonable amount of effort to implement in a VHDL-based fault simulator.

One possible way to overcome the difficulty associated with implementing hierarchical concurrent fault simulation is to use a standard high-level language to perform the fault list management functions. For example, having a C program to perform the fault list propagation algorithm is one possible solution. The C fault list propagation algorithm interfaces to a VHDL model via a Program Language Interface (PLI). Unfortunately, the PLI is presently not standardized. A PLI standardization effort is currently in progress by an IEEE working group. The lack of a PLI standard means that any C based approach to perform fault list propagation is VHDL simulator specific. More research is required to determine the feasibility of using a high-level language to perform fault list propagation.

The basic high-level structure of VHDL-based fault simulation is introduced to conclude the discussion on VHDL-based fault simulation. A structural diagram of VHDL-based fault simulation is included as Figure 10.1. There are several pieces of information which must be provided by the designer before fault simulation can be performed. The designer supplied information is as follows: (1) VHDL model of the DUT, (2) fault list associated with DUT, (3) the inputs, and (4) the

known outputs associated with each input. A VHDL-based standard known as the Waveform And Vector Exchange Specification (WAVES) can be used to store the inputs and outputs for the DUT [88]. The VHDL simulator and fault injector blocks in Figure 10.1 are responsible for fault simulating the DUT. The simulated outputs are then compared with the stored outputs with a comparator to determine if the injected fault is detected. The output of the comparison is stored as a result. Post and pre processing blocks are included in Figure 10.1 to represent the use of fault equivalency. The preprocessing step entails the determination of device-level equivalent faults. The device-level equivalent fault information is then used to reduce the size of the fault list before fault simulation is begun. The post processing block represents the use of partial circuit fault equivalency with techniques such as CPT. The partial circuit fault equivalence information is then used to remove equiv-
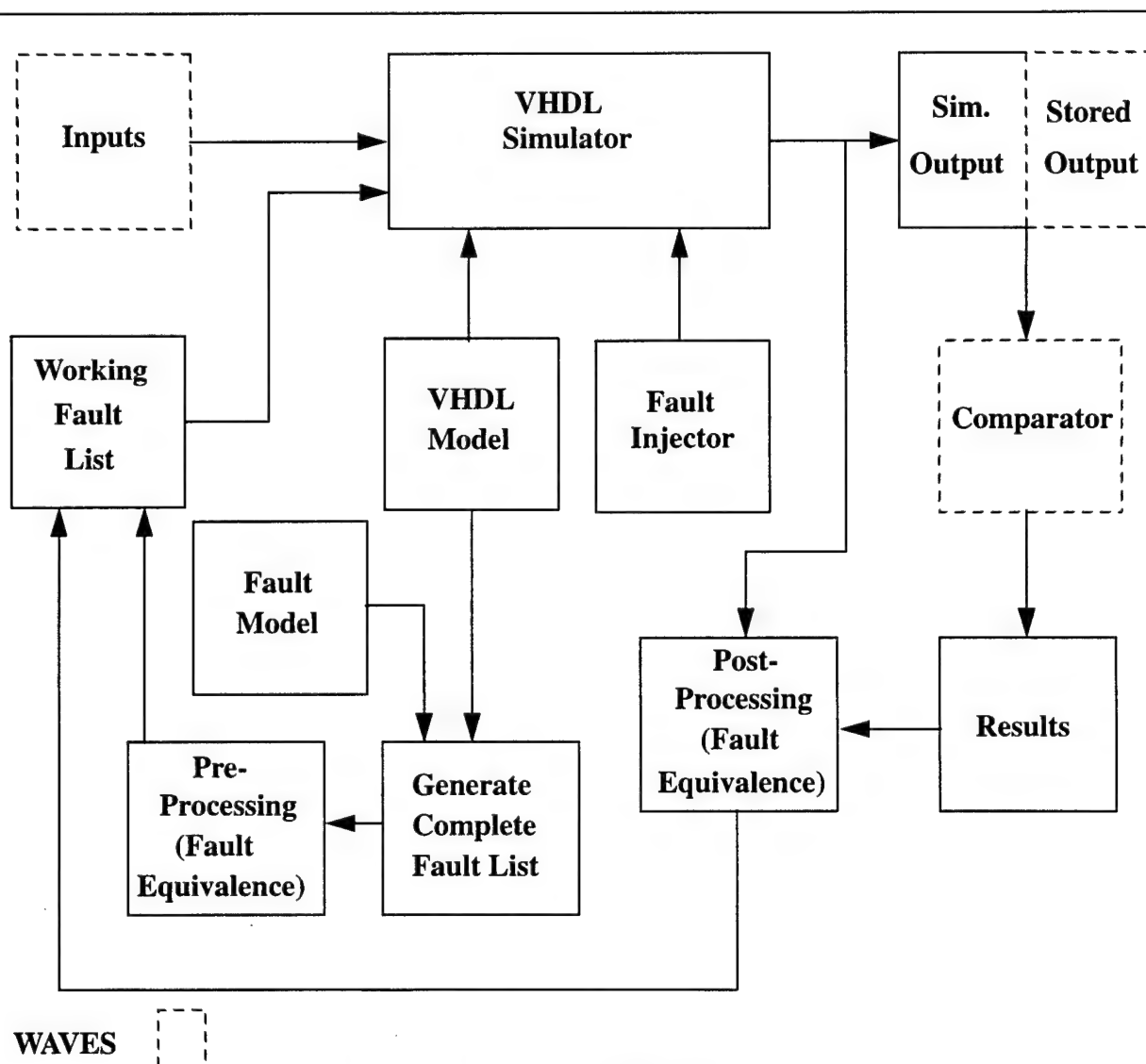

Figure 10.1.   Block diagram of a VHDL-based fault simulator.

alent detected faults from the fault list to reduce the remaining number of required fault simulations.

11. Conclusion

Fault simulation is currently an integral part of the design process for digital components. The primary purpose of fault simulation is to evaluate the effectiveness of the design for test methodology for a given device. One metric which is used to measure the effectiveness of a given test strategy is the fault coverage of the DUT. The data provided by fault simulation is used to generate the fault coverage estimate. The closer the fault coverage estimate is to 1.0 the better the test strategy and the higher the probability that manufacturing defects in the DUT will be detected when evaluated by the test strategy.

This report provides a survey of fault simulation techniques. An overview of uniprocessor fault simulation techniques is presented in Section 5. The overview describes nine different uniprocessor fault simulation techniques along with a survey on VHDL-based fault simulation. Parallel processor fault simulation techniques are presented in Section 6. A description of hardware accelerators and their use in fault simulation is provided in Section 7. An overview of fault grading techniques is included as Section 8. Likewise, an overview of test pattern generation techniques is presented in Section 9. The applicability of existing fault simulation techniques described in Sections 5 and 6 for VHDL-based fault simulation is described in Section 10.

The end use of this report is to provide a starting point for the development of new and novel VHDL-based fault simulation techniques. The ultimate objective is to develop a VHDL fault simulation technique that can use any existing VHDL model. Ideally, the fault simulation technique would treat the VHDL model as a black box. Specifically, there would be no constraint placed on the designer in constructing a VHDL model so long as the model adhered to the VHDL standard. The fault simulation is performed by injecting a fault into the VHDL model and using any compliant VHDL simulator to simulate the faulty DUT. The data generated by the fault simulation is then analyzed by an external program to perform bookkeeping functions such as: (1) fault dropping, (2) fault collapsing due to fault equivalence, and (3) fault coverage estimation. The ideal fault simulation technique would allow for fault simulation of both structural and behavioral VHDL models using a unified methodology.

Bibliography

[1]   Abramovici, M., M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, AT&T Bell Laboratories and W. H. Freeman & Company, 1990.

[2]   Abramovici, M., M. A. Breuer, and K. Kumar, "Concurrent Fault Simulation and Functional Level Modeling," *14th Design Automation Conference Proceedings*, June 1977, pp. 128-135.

[3]   Abramovici, M., B. Krishnamurthy, R. Mathews, B. Rogers, M. Schulz, S. Seth, and J. Waicukauski, "What is the Path to Fast Fault Simulation? (A Panel Discussion)", *International Test Conference Proceedings*, September 1988, pp. 183-192.

[4]   Abramovici, M., J.J. Kulikowski, P. R. Menon, and D. T. Miller, "SMART and FAST: Test Generation for VLSI Scan-Design Circuits," *IEEEE Design & Test of Computers*, Vol. 3, No. 4, pp. 43-54.

[5]   Abramovici, M. and P. R. Menon, "A Practical Approach to Fault Simulation and Test Generation for Bridging Faults," *IEEE Transactions on Computers*, Vol C-34, No. 7, July 1985, pp. 658-663.

[6]   Abramovici, M, P. R. Menon, D. T. Miller, "Critical Path Tracing- An Alternative to Fault Simulation", *20th Design Automation Conference Proceedings*, 1983, pp. 214-220.

[7]   Abramovici M., P. R. Menon, and D. T. Miller, "Critical Path Tracing: an Alternative to Fault Simulation," *IEEE Design and Test of Computers*, Vol. 1, No. 1, February 1984, pp. 83-93.

[8]   Agrawal, P., V. D. Agrawal, K.-T. Cheng, and R. Tutundjian, "Fault Simulation in a Pipelined Multiprocessor System ", *International Test Conference Proceedings*, August 1989, pp. 727-734.

[9]   Agrawal, P., W. J. Dally, W. C. Fisher, H. V. Jagadish, A. S. Krishakumar, and R. Tutundjian, "MARS: A Multiprocessor-Based Programmable Accelerator," *IEEE Design & Test of Computers*, Vol. 4, No. 5, October 1987, pp. 28-36.

[10]   Agrawal, V. D., "An Information Theoretic Approach to Digital Fault Testing," *IEEE Transactions on Computers*, Vol., C-30, No. 8, August 1981, pp. 582-587.

[11]   Agrawal, V. D., "Sampling technique for determining fault coverage in LSI circuits," *Journal of Digital Systems*, Vol V, No. 3, 1981, pp. 189-202.

[12] Agrawal, V. D., S. C. Seth, and P. Agrawal, "LSI Product Quality and Fault Coverage," *18th Design Automation Conference Proceedings*, June 1981, pp. 196-203.

[13] Airapetian, A. N, and J. F. McDonald, "Improved Test Set Generation Algorithm for Combinational Logic Control," *Digest of Papers 9th Annual International Symposium on Fault-Tolerant Computing*, June 1979, pp. 133-136.

[14] Akers, S. B., "A Logic System for Fault Test Generation," *IEEE Transactions on Computers*, Vol. C-25, No. 6, June 1976, pp. 594-604.

[15] Akers, S. B., B. Krishnamurthy, S. Park, and A. Swaminathan, "Why Is Less Information from Logic Simulation more Useful in Fault Simulation?", *International Test Conference Proceedings*, September 1990, pp. 786-800.

[16] Antreich, K. J. and M. H. Schulz, "Fast fault simulation for scan based VLSI logic," *European Conference on Circuit Theory and Design Proceedings*, 1987.

[17] Antreich, K. J. and M. H. Schulz, "Fast Fault Simulation in Combinational Circuits", *IEEE Int. Conference on CAD, ICCAD-86*, Nov. 1986, pp. 330-333.

[18] Antreich, K. J. and M. H. Schulz, "Accelerated Fault Simulation and Fault Grading in Combinational Circuits, *IEEE Transactions on CAD*, Vol. CAD-6, No. 5, September 1987, pp. 704-712.

[19] Armstrong, D. B., "A Deductive method of Simulating Faults in Logic Circuits," *IEEE Transactions on Computers*, Vol. C-22, May 1972, pp. 464-477.

[20] Attest Software, Inc., "Fault Simulation Automatic Test Pattern Generation Design for Testability", Users Manual, 1995.

[21] Bardell, P. H, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, New York, 1987.

[22] Barzilai, Z., D. K. Beece, L. M. Huisman, V. S. Iyengar, and G. M. Silberman, "SLS-A Fast Switch Level Simulator for Verification and Fault Coverage Analysis," *Design Automation Conference Proceedings*, 1986, pp. 164-170.

[23] Barzilai, Z., J. L. Carter, V. S. Iyengar, I. Nair, B. K. Rosen, J. Rutledge, and G. M. Silberman, "Efficient Fault Simulation of CMOS Circuits with Accurate Models", *International Test Conference Proceedings*, September 1986, pp. 520-529.

[24] Barzilai, Z., J. L. Carter, B. K. Rosen, and J. D. Rutledge, "HSS -- A High Speed Simulator," IBM Research Report RC 11738, Thomas J. Watson Research Center, Yorktown Heights, NY, March 1986.

[25] Bataineh, A., F. Ozguner, and I. Szauter, "Parallel Logic and Fault Simulation Algorithms for Shared Memory Vector Machines," *ICCAD-92 Proceedings*, November 1992, pp. 369-372.

[26] Becker, B., R. Hahn, R. Krieger, and U. Sparmann, "Structure Based Methods for Parallel Pattern Fault Simulation in Combinational Circuits," *European Design Automation Conference Proceedings*, 1991, pp. 1929-1934.

[27] Becker, B. and R. Krieger, "FAST-SC: Fast fault simulation in synchronous sequential circuits," *VLSI Design Conference Proceedings*, 1993, pp. 40-45.

[28] Benmehrez, C., and J. F. McDonald, "Measured Performance of a Programmed Implementation of the Subscripted D-Algorithm," *20th Design Automation Conference Proceedings*, June 1983, pp. 308-315.

[29] Blaauw, D. T., D. G. Saab, P. Banerjee, and J. A. Abraham, "Functional Abstraction of Logic Gates for Switch-Level Simulation," *EDAC Proceedings*, February 1991, pp. 329-323.

[30] Blank, T., "A Survey of Hardware Accelerator used in Computer-Aided Design," *IEEE Design and Test of Computers*, Vol. 1, No. 3, August 1984, pp. 21-39.

[31] Bose, S. and P. Agrawal, "Concurrent Fault Simulation for Logic Gates and Memory Blocks of Message Passing Multicomputers", *29th Design Automation Conference Proceedings*, June 1992, pp. 332-335.

[32] Bose, A., P. Kozak, C.-Y. Lo, H. N. Nham, E. Pacas-Skewes, and K. Wu, "A Fault Simulator for MOS LSI Circuits," *19th Design Automation Conference Proceedings*, July 1982, pp. 400-409.

[33] Breuer, M. A., *Diagnosis and Reliable Design of Digital Systems*, Potomac, Md., Computer Science Press, 1976.

[34] Brglez, F., "A Fast Fault Grader: Analysis And Applications," *International Test Conference Proceedings*, November 1985, pp 785-794.

[35] Brglez, F., P. Pownall, and R. Hum, "Accelerated ATPG and fault grading via testability analysis," *Proceedings of IEEE International Symposium on Circuits and Systems*, June 1985, pp. 695-698.

[36] Briers, A. J. and K. A. E. Totton, "Random Pattern Testability by Fast Fault Simulation ", *International Test Conference Proceedings*, September 1986, pp. 274-281.

[37] Bryant, R. E. "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol C-35, No. 8, August 1986.

[38] Bryant, R. E., "Symbolic Simulation -- Techniques and Applications," *27th Design Automation Conference Proceedings*," June 1990, pp. 517-521.

[39] Bryant, R. E. and M. D. Schuster, "Fault Simulation of MOS Digital Circuits," *VLSI Design*, October 1983, pp. 24-30.

[40] Bryant, R. E. and M. D. Shuster, "Performance Evaluation of FMOSSIM, a Concurrent Switch-Level Fault Simulator," *Design Automation Conference Proceedings*, 1985, pp 715-719.

[41] Carter, J. L., S. F. Dennis, V. S. Iyengar, and B. K. Rosen, "ATPG via Random Pattern Simulation," *Proceedings of the International Conference on Computer Design*, 1988, pp. 334-337.

[42] Carter, J. L., V. S. Iyengar, and B. K. Rosen, "Efficient Test Coverage Determination for Delay Faults ", *International Test Conference Proceedings*, September 1987, pp. 418-427.

[43] Caisso, J.-P. and B. Courtois, "Fault Simulation and Test Pattern Generation at the Multiple-Valued Switch Level," *IEEE International Test Conference Proceedings*, 1988, pp. 94-101.

[44] Center for Digital Systems Engineering, Research Triangle Institute, "Mission-Critical Failure Effects Analysis Using Quantitative Techniques," Final Report, Contract No. F30602-93-C-0009, Rome Laboratory/ERM, Griffiss AFB, NY 13441-4514, April 1994.

[45] Cha, C. W., W. E. Donath, and F. Ozguner, "9-V Algorithm for Pattern Generation of Combination Digital Circuits," *IEEE Transactions on Computers*, Vol. C-27, No. 3, March 1978, pp. 193-200.

[46] Chan, T. and E. Law, "MegaFAULT: A Mixed-Mode, Hardware Accelerated Concurrent Fault Simulator," *International Conference on Computer-Aided Design*, November 1986, pp. 394-397.

[47] Chang, H. P. and J. A. Abraham, "Use of High Level Descriptions for Speedup of Fault Simulation", *International Test Conference Proceedings*, September 1987, pp. 278-285.

[48] Chang, H. Y.-P., S. G. Chappel, C. H. Elmendorf, and L. D. Schmidt, "Comparison of Parallel and Deductive Fault Simulation Methods," *IEEE Transactions on Computers*, Vol. C-23, No. 11, November 1974, pp. 1132-1138.

[49] Chappell, S. G., C. H. Elmendorf, and L. D. Schmidt, "LAMP: Logic Circuit Simulators," *Bell System Technical Journal*, Vol. 53, No. 8, June 1974, pp. 1431-1449.

[50] Chen, J. E., C. L. Lee, and W. Z. Shen, "Single-Fault Fault Collapsing Analysis in Sequential Logic Circuits", *International Test Conference Proceedings*, September 1990, pp. 809-814.

[51] Cheng, W.-T. and M.-L. Yu, "Differential Fault Simulation - A Fast Method Using Minimal Memory", *26th Design Automation Conference Proceedings*, June 1989, pp. 424-428.

[52] Cheng, W., and J. H. Patel, "PROOFS: A Super Fast Fault Simulator for Sequential Circuits," *European Design Automation Conference Proceedings*, 1990, pp. 475-479.

[53] Cho, K. and R. E. Bryant, "Test pattern generation for sequential MOS circuits by symbolic fault simulation," *26th Design Automation Conference Proceedings*, June 1989, pp. 418-423.

[54] Cleverley, D. S., "The Role of Testing in Achieving Zero Defects," *International Test Conference Proceedings*, October 1983, pp. 248-253.

[55] d'Abreu, M. A., and E. W. Thompson, d'Abreu, M. A., and E. W. Thompson, "An Accurate Functional Level Concurrent Fault Simulator," *17th Design Automation Conference Proceedings*, June 1980, pp. 210-217.

[56] Dalpasso, M., M. Favalli, P. Olivo, and B. Ricco, "Switch-Level Fault Simulation by Critical Path Tracing," *2nd ETC Proceedings*, April 1991, pp. 181-190.

[57] Daehn, W. and M. Geilert, "Fast Fault Simulation for Combinational Circuits by Compiler Driven Single Fault Propagation", *International Test Conference Proceedings*, September 1987, pp. 286-292.

[58] Daehn, W., D. Kannemacker, and J. Castagne, "Vector Length Control for Compiled Code Event Driven Pattern Parallel Fault Simulation," *2nd European Test Conference*, April 1991, pp. 165-170.

[59] Daoud, R. and F. Ozguner, "Highly Vectorizable Fault Simulation on the Cray X-MP Supercomputer," *IEEE Transactions on Computer-Aided Design*, December 1989, pp. 1362-1365.

[60] Davidson, S., "Fault Simulation at the Architectural Level," *International Test Conference Proceedings*," Washington D. C., September 1984, pp. 669-679.

[61] Davidson, S. and J. L. Lewandowski, "ESIM/AFS - A Concurrent Architectural Level Fault Simulator", *International Test Conference Proceedings*, September 1986, pp. 375-383.

[62] Debany, W. H., C. R. P. Hartmann, P. K. Varshney, and K. G. Mehrotra, "Comparison of Random Test Vector Generation Strategies," *IEEE International Conference on Computer-Aided Design*, November 1991, pp. 244-247.

[63] Debany, W. H., P. K. Varshney, and C. R. P. Hartman, "Random Test Length With and Without Replacement," *Electronics Letters*, Vol. 22, No. 20, September 1986, pp. 1074-1075.

[64] DeLong, T. A, B. W. Johnson, and J. A. Profeta, III, "A Fault Injection Technique for VHDL Behavioral-Level Models," *IEEE Design and Test*, submitted for publication.

[65] Denneau, M. M., "The Yorktown Simulation Engine," *19th Design Automation Conference Proceedings*, June 1982, pp. 55-59.

[66] Duba, P. A., R. K. Roy, J. A. Abraham, and W. A. Rogers, "Fault Simulation in a Distributed Environment ", *25th Design Automation Conference Proceedings*, June 1988, pp. 686-691.

[67] Eldred, R. D., "Test Routines Based on Symbolic Logic Statements," Journal of the ACM, Vol. 6, No. 1, January 1959, pp. 33-36.

[68] Friedman, M., D. Harel, F. Maamari, and J. Rajski, "A Dominators View of Stem Regions in Combinational Logic and its Application to Fault Simulation," Technical Report 87-50, Computer Research Laboratory, Tektronix Laboratories, 1987.

[69] Fujiwara, H. and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Transactions on Computers*, Vol. C-32, No. 12, December 1983, pp. 1137-1144.

[70] Funatsu, S., M. Takahashi, and M. Shibata, "Digital Fault Simulation in Bidirectional Bus Circuits Environments," *10th Fault Tolerant Computing Symposium Proceedings*, October 1980, pp. 71-78.

[71] Gaede, R. K., M. R. Mercer, K. M. Butler, and D. E. Ross, "CATAPULT: Concurrent Automatic Testing Allowing Parallelization and Using Limited Topology," *25th Design Automation Proceedings*, June 1986, pp. 597-600.

[72] Gai, S., P. L. Montessoro, and F. Somenzi, "MOZART: A Concurrent Multilevel Simulator," *IEEE Transactions on Computer Aided Design*, Vol. CAD-7, No. 9, September 1988, pp. 1005-1016.

[73] Gai, S., P. L. Montessoro, and F. Somenzi, "The Performance of the Concurrent Fault Simulation Algorithms in MOZART", *25th Design Automation Conference Proceedings*, June 1988, pp. 692-697.

[74] Gai, S., F. Somenzi, and E. Ulrich, "Advanced Techniques for Concurrent Multilevel Simulation," *IEEE ICCAD-86 Proceedings*, November 1986, pp. 334-337.

[75] Ganapathy, G. and J. A. Abraham, "Hardware Acceleration Alone Will Not Make Fault Grading ULSI a Reality", *International Test Conference Proceedings*, October 1991, pp. 848-857.

[76] Ghosh, S., "A Distributed Algorithm for Fault Simulation of Combinatorial and Asynchronous Sequential Digital Designs, Utilizing Circuit Partitioning, on Loosely-Coupled Parallel Processors", *Microelectronic Reliability*, Vol. 35, No. 6, 1995, pp. 947-967.

[77] Ghosh, S. "Behavior-Level Fault Simulation," *IEEE Design and Test*, June 1988, pp 31-42.

[78] Giraldi, J. and M. L. Bushnell, "EST: The New Frontier in Automatic Test-Pattern Generation," *27th Design Automation Conference Proceedings*, June 1990, pp. 667-672.

[79] Goel, P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, Vol. C-30, No. 3, March 1981, pp. 215-222.

[80] Goel, P., "Test Generation Costs Analysis and Projections," *17th Design Automation Conference Proceedings*, June 1980, pp. 77-84.

[81] Goel, P., C. L. Huang, and R. E. Blauth, "Application of Parallel Processing to Fault Simulation," *Proceedings of the IEEE Conference on Parallel Processing*, 1986, pp. 785-788.

[82] Goel, P. and P. R. Moorby, "Fault-Simulation Techniques for VLSI Circuits," VLSI Design, July 1984, pp. 22-26.

[83] Goel, P. and B. C. Rosales, "PODEM-X: An Automatic Test Generation System for VLSI Logic Structures," *18th Design Automation Conference Proceedings*, June 1981, pp. 260-268.

[84] Goel, P., H. Licha, T. E. Rosser, T. J. Stroh, and E. B. Eichelberger, "LSSD Fault Simulation Using Conjunctive Combinational and Sequential Methods," *1980 Test Conference Proceedings*, Nov. 1980, pp 371-376.

[85] Goldstein, L. H., "Controllability/observability analysis of digital circuits," *IEEE Transactions on Circuits and Systems*, Vol. 26, No. 9, 1979, pp. 685-693.

[86] Gouders, N. and R. Kaibel, "PARIS: A Parallel Pattern Fault Simulator for Synchronous Sequential Circuits," *International Conference on CAD Proceedings*, November 1991, pp. 542-545.

[87] Guzolek, J. F., W. A. Rogers, and J. A. Abraham, "WRAP: An Algorithm for Hierarchical Compression of Fault Simulation Primitives," *IEEE ICCAD-86 Proceedings*, November 1986, pp. 338-341.

[88] Hanna, J. P., "Using WAVES for VHDL Model Verification ", *VIUF Conference Proceedings*, April 1995, pp. 6.11-6.21.

[89] Hardie, F. H and R. J. Suhocki, "Design and use of fault simulation for Saturn computer design," *Wescon Tech. Paper*, Vol. 10, pp. 4, August 1966, pp. 1-22.

[90] Harel, D., "A Graph Compaction Approach to Fault Simulation ", *25th Design Automation Conference Proceedings*, June 1988, pp. 601-604.

[91] Harel, D. and B. Krishnamurthy, "Is There Hope for Linear Time Fault Simulation?," *17th Fault Tolerant Computing Symposium Proceedings*, 1987, pp. 28-33.

[92] Harel, D., R. Sheng, and J. Udell, "Efficient Single Fault Propagation in Combinational Circuits," *ICCAD-87 Proceedings*, November 1987, pp. 2-5.

[93] Hayes, J. P., "A Fault Simulation Methodology for VLSI," *19th Design Automation Conference*, June 1982, pp. 393-399.

[94] Hayes, J. P., "Fault Modeling ", *IEEE Design and Test of Computers*, Vol. 2, No. 2, April 1985, pp. 88-95.

[95] Henckels, L. P., K. M. Brown, and C. Lo, "Functional Level, Concurrent Fault Simulation," *1980 IEEE Test Conference Proceedings*, 1980, pp. 479-485.

[96] Hirose, F., N. Kawato, M. Ishii, H. Hamamura, J. Niitsuma, K. Uchida, T. Shindo, and H. Yamada, "Simulation Processor "SP"," *IEEE ICCAD-87 Proceedings*, November 1987, pp. 484-487.

[97] Hirose, F., K. Takayama, and N. Kawato, "A Method to Generate Tests for Combinational Logic Circuits using an Ultrahigh-speed Logic Simulator", *International Test Conference Proceedings*, September 1988, pp. 102-107.

[98] Hong, S. J., "Fault Simulation Strategy for Combinational Logic Networks", *Proc. 8th Int. Symposium on Fault Tolerant Computing*, 1978, pp. 96-99.

[99] Hsiao, M. S. and J. H. Patel, "A new architectural-level fault simulation using propagation prediction of grouped fault-effects," *IEEE International Conference on Computer Design Proceedings*, October 1995.

[100] Huisman, L. M. and R. Daoud, "Fault Simulation of Logic Designs on Parallel Processors with Distributed Memory", *International Test Conference Proceedings*, September 1990, pp. 690-696.

[101] Huisman, L., I. Nair, and R. Daoud, "Fault Simulation on Message Passing Parallel Processors," *Distributed Memory Computing Conference Proceedings*, April 1990.

[102] Huisman, L., I. Nair, and R. Daoud, "Fault Simulation on Message Passing Parallel Processors," *International Conference on Computer Aided Design*, November 1990, pp. 33-41.

[103] Hwang, T.-S., C. L. Lee, W. Z. Shen, and C. P. Wu, "A Parallel Pattern Mixed-Level Fault Simulator ", *27th Design Automation Conference Proceedings*, June 1990, pp. 716-719.

[104] Ikos Systems, Inc., "Voyager FS - VHDL Mixed-Level Fault Simulator," Product Literature, November 1995.

[105] Ishiura, N., M. Ito, and S. Yajima, "Dynamic 2-Dimensional Parallel Simulation Technique for High Speed Fault Simulation on a Vector Processor," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 8, 1990, pp. 868-875.

[106] Ishiura, N., M. Ito, and S. Yajima, "High-Speed Fault Simulation Using a Vector Processor Design," *International Conference on Computer Aided Design Proceedings*, November 1987, pp. 10-13.

[107] Iyengar, V. S. and D. T. Tang, "On Simulating Faults in Parallel," *Digest of Papers 18th International Symposium on Fault-Tolerant Computing*, June 1988, pp. 110-115.

[108] Jain, S. K. and V. D. Agrawal, "STAFAN: An Alternative to Fault Simulation," *21st Design Automation Conference Proceedings*, 1984, pp. 18-23.

[109] Jain, S. K. and V. D. Agrawal, "Statistical Fault Analysis," *IEEE Design and Test*, Vol.2, February 1985, pp. 38-44.

[110] Jenn, E., J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool, "*24th Fault Tolerant Computing Symposium Proceedings*, June 1994, pp. 66-75.

[111] Johnson, B. W., *Design and Analysis of Fault Tolerant Systems*, Addison-Wesley, Reading, Massachusetts, 1989.

[112] Kawai, M. and J. P. Hayes, "An Experimental MOS Fault Simulation Program CSASIM, "*21st Design Automation Conference Proceedings*, June 1984, pp. 2-9.

[113] Ke, W., S. Seth, and B. B. Bhattacharya, "A Fast Fault Simulation Algorithm for Combinational Circuits," *International Conference on Computer Aided Design Proceedings*, November 1988, pp. 166-169.

[114] Kirkland, T., and M. R. Mercer, "A topological Search Algorithm for ATPG," *24th Design Automation Conference Proceedings*, June 1987, pp. 502-508.

[115] Kitamura, Y., "Exact Critical Path Tracing Fault Simulation on Massively Parallel Processor AAP2," *International Conference on Computer Aided Design*, November 1989, pp. 474-477.

[116] Kitamura, Y., "Sequential Circuit Fault Simulation by Fault Information Tracing Algorithm: FIT", *28th Design Automation Conference Proceedings*, June 1991, pp. 151-154.

[117] Kjelkerud, E., and O. Thessen, "Techniques for Generalized Deductive Fault Simulation," *Journal of Design Automation and Fault Tolerant Computing*, Vol. 1, October 1977, pp. 377-390.

[118] Koeppe, S., "Modeling and Simulation of Delay Faults in CMOS Logic Circuits", *International Test Conference Proceedings*, September 1986, pp. 530-536.

[119] Kohavi, Z., *Switching and Finite Automata Theory*, McGraw-Hill Inc., New York, New York, 1978.

[120] Krieger, R., B. Becker, and M. Keim, "A Hybrid Fault Simulator for Synchronous Sequential Circuits", *International Test Conference Proceedings*, October 1994, pp. 614-623.

[121] Krieger, R., B. Becker, and M. Keim, "Symbolic Fault Simulation for Sequential Circuits and the Multiple Observation Time Test Strategy", *32nd Design Automation Conference Proceedings*, June 1995, pp. 339-344.

[122] Kubiak, K. and W. K. Fuchs, "Multiple-Fault Simulation and Coverage of Deterministic Single-Fault Test Sets", *International Test Conference Proceedings*, October 1991, pp. 956-962.

[123] Kung, C. and C. Lin, "Parallel sequence fault simulation for synchronous sequential circuits," *European Conference on Design Automation Proceedings*, 1992, pp. 434-438.

[124] Larrabee, T., "Efficient Generation of Test Patterns Using Boolean Difference," *International Test Conference Proceedings*, August 1989, pp. 795-801.

[125] Lee, D. H. and S. M. Reddy, "On Efficient Concurrent Fault Simulation for Synchronous Sequential Circuits", *29th Design Automation Conference Proceedings*, June 1992, pp. 327-331.

[126] Lee, H. K. and D. S. Ha, "An Efficient, Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation", *International Test Conference Proceedings*, October 1991, pp. 946-954.

[127] Lee, H. L. and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits", *29th Design Automation Conference Proceedings*, Anaheim, California, June 1992, pp. 336-340.

[128] Levendel, Y. H., and P. R. Menon, "Comparison of Fault Simulation Methods -- Treatment of Unknown Signal Values," *Journal of Digital Systems*, Vol. 4, No. 4, Winter 1980, pp. 443-459.

[129] Levendel, Y. H., and P. R. Menon, "Fault-simulation methods -- Extensions and comparison," *Bell System Technical Journal*, Vol. 60, Nov. 1981, pp 2235-2259.

[130] Levendel, Y. H, P. R. Menon, and S. H. Patel, "Parallel Fault Simulation Using Distributed Processing," *The Bell System Technical Journal*, Vol. 62, No. 10, December 1983, pp. 3107-3129.

[131] Li., Y.-L. and C.-W. Wu, "Cellular Automata for Efficient Parallel Logic and Fault Simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 6, June 1995, pp. 740-749.

[132] Lo C. Y., H. N. Nham, and A. K. Bose, "Algorithms for an Advanced Fault Simulation System in Motis," *IEEE Transactions on Computer Aided Design*, March 1987, pp 210-218.

[133] Maamari, F. and J. Rajski, "A Fault Simulation Method Based on Stem Regions," *ICCAD-88 Proceedings*, November 1988, pp. 170-173.

[134] Maamari, F. and J. Rajski, "A Method of Fault Simulation Based on Stem Regions," *IEEE Transactions on Computer Aided Design*, Vol. 9, No. 2, February 1990, pp. 212-220.

[135] Maamari, F., and J. Rajski, "A Reconvergent Fanout Analysis for Efficient Exact Fault Simulation of Combinational Circuits," *18th Fault Tolerant Computing Symposium Proceedings*, June 1988, pp. 122-127.

[136] Maamari, F. and J. Rajski, "The Dynamic Reduction of Fault Simulation ", *International Test Conference Proceedings*, September 1990, pp. 801-808.

[137] Machlin, D., D. Gross, S. Kadkade, and E. Ulrich, "Switch-Level Concurrent Fault Simulation Based On a General Purpose List Traversal Mechanism", *International Test Conference Proceedings*, September 1988, pp. 574-581.

[138] Mahlstedt, U. and J. Alt, "Simulation of non-classical Faults on the Gate Level - The Fault Simulator COMSIM -", *International Test Conference Proceedings*, October 1993, pp. 883-892.

[139] Markas, T., M. Royals, and N. Kanopoulos, "On Distributed Fault Simulation," *IEEE Computer*, Vol. 23, No. 1, January 1990, pp. 40-52.

[140] McCluskey, E. J, *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice Hall, Englewood Hills, California, 1986.

[141] Menon, P. R., Y. Levendel, and M. Abramovici, "Critical Path Tracing in Sequential Circuits," *ICCAD-88 Proceedings*, November 1988, pp. 162-165.

[142] Menon, P. R. and S. G. Chappel, "Deductive Fault Simulation with Functional Blocks," *IEEE Transactions on Computers*, Vol C-27, August 1978, pp. 687-695.

[143] Meyer, W. and R. Camposano, "Fast, Accurate, Integrated Gate- and Switch-Level Fault Simulation," *IEEE European Test Conference Proceedings*, 1993.

[144] Meyer, W. and R. Camposano, "Fast Hierarchical Multi-Level Fault Simulation of Sequential Circuits with Switch-Level Accuracy", *30th Design Automation Conference Proceedings*, June 1993, pp. 515-519.

[145] MIL-STD-883D, "Test Methods and Procedures for Microelectronics," Method 5012.1, November 1991.

[146] Mine, B., "Put the Pedal to the Metal with Simulation Accelerators," *Electronic Design*, September 1987, pp. 39-52.

[147] Montessoro, P. L. and S. Gai, "Creator: General and Efficient Multilevel Concurrent Fault Simulation", *28th Design Automation Conference Proceedings*, June 1991, pp. 160-163.

[148] Motohara, A. M. Murakami, M. Urano, Y. Masuda, and M. Sugano, "An Approach to Fast Hierarchical Fault Simulation ", *25th Design Automation Conference Proceedings*, June 1988, pp. 698-703.

[149] Motohara, A. K. Nishimura, H. Fujiwara, and I. Shirakawa, "A Parallel Scheme for Test Pattern Generation," *International Conference on Computer Aided Design Proceedings*, Nov. 1986, pp. 156-159.

[150] Mueller-Thuns, R. B., D. G. Saab, R. F. Damiano, and J. A. Abraham, "Portable Parallel Logic and Fault Simulation," *International Conference on Computer Aided Design*, November 1989, pp. 506-509.

[151] Muth, P., "A Nine-Valued Circuit Model for Test Generation," *IEEE Transactions on Computers*, Vol. C-25, No. 6, June 1976, pp. 630-636.

[152] Nagumo, T., M. Nagai, T. Nishida, M. Miyoshi, and S. Miyamoto, "VFSIM: Vectorized fault simulator using a reduction technique excluding temporarily unobservable faults", *31st Design Automation Conference Proceedings*, June 1994, pp. 510-515.

[153] Narayanan, V. and V. Pitchurmani, "A Massively Parallel Algorithm for Fault Simulation on the Connection Machine", *26th Design Automation Conference Proceedings*, June 1989, pp. 734-737.

[154] Narayanan, V. and V. Pitchumani, "A Parallel Algorithm for Fault Simulation on the Connection Machine", *International Test Conference Proceedings*, September 1988, pp. 89-93.

[155] Narayanan, V. and V. Pitchumani, "Fault Simulation on massively parallel SIMD machines: Algorithms, implementations and results," *Journal Electronic Testing: Theory and Application*, Vol. 3, Sept. 1992, pp. 79-92.

[156] Navabi, S., N. Cooray, and R. Liyanage, "Modeling for Fault Insertion and Parallel Fault Simulation", *VIUF Conference Proceedings*, April 1993, pp. 75-89.

[157] Navabi, Z. and M. Shadfar, "A VHDL Based Test Environment Including Models for Equivalence Fault Collapsing", *VHDL International Users' Forum*, May 1994, pp. 18-25.

[158] Nicholls, W. H., A. W. Nordsieck, and M. Soma, "Experimental evaluation of concurrent fault simulation algorithms on scalable, hierarchically defined test cases", *International Test Conference Proceedings*, September 1990, pp. 698-705.

[159] Nicholls, W. H. and M. Soma, "Fault bundling: reducing machine evaluation activity in hierarchical concurrent fault simulation", *International Test Conference Proceedings*, September 1988, pp. 569-573.

[160] Niermann, T. M., W.-T. Cheng, and J. H. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator", *27th Design Automation Conference Proceedings*, June 1990, pp. 535-540.

[161] Nishida, T., Miyamoto, S., T. Kozawa, and K. Satoh, "RFSIM: Reduced Fault Simulation," *IEEE Transactions on Computer Aided Design*, Vol. CAD-6, No. 3, May 1987, pp. 392-403.

[162] Ostapko, D. L., Z. Barzilai, and G. M. Silberman, "Fast Fault Simulation in a Parallel Processing Environment", *International Test Conference Proceedings*, September 1987, pp. 293-298.

[163] Ozguner, F., W. E. Donath, and C. W. Cha, "On Fault Simulation Techniques," *Journal of Design Automation and Fault Tolerant Computing*, Vol. 3, April 1979, pp. 83-92.

[164] Ozguner, F. and R. Daoud, "Vectorized fault simulation on the Cray-XP supercomputer," *ICCAD Proceedings*, 1988, pp. 198-201.

[165] Ozguner, F., C. Aykanat, and O. Khalid, "Logic Fault Simulation on a Vector Hypercube Multiprocessor," *3rd Conference on Hypercube Concurrent Computers and Applications Proceedings*, January 1988, pp. 1108-1116.

[166] Pfister, G. F., "The Yorktown Simulation Engine," *Design Automation Conference Proceedings*, 1982, pp. 31-54.

[167] Pfister, G. F., "The Yorktown Simulation Engine," *Proceedings of the IEEE*, Vol. 74, No. 6, June 1986, pp. 850-860.

[168] Pitchumani, V., P. Mayor, and N. Radia, "A System for Fault Diagnosis and Simulation of VHDL descriptions", *28th Design Automation Conference Proceedings*, June 1991, pp. 144-150.

[169] Pitchumani, V., P. Mayor, and N. Radia, "Fault Diagnosis of VHDL descriptions using Functional Fault Models", *VHDL Users' Group Conference*, April 1991, pp. 29-33.

[170] Pitchumani, V., P. Mayor, and N. Radia, "Fault Diagnosis using Functional Fault Models for VHDL descriptions", *International Test Conference Proceedings*, October 1991, pp. 327-337.

[171] Pomeranz, I., S. Reddy, and L. Reddy, "Increasing fault coverage for synchronous sequential circuits by the multiple observation time test strategy," *International Conference on CAD Proceedings*, 1991, pp. 454-457.

[172] Pomeranz, I. and S. M. Reddy, "On Fault Simulation for Synchronous Sequential Circuits," *IEEE Transactions on Computers*, Vol. 44, No. 2, February 1995, pp. 335-340.

[173] Pomeranz, I. and S. M. Reddy, "Test Generation for synchronous sequential circuits using multiple observation times," *International Symposium on Fault Tolerant Computer Proceedings*, 1991, pp. 52-59.

[174] Pradhan, D. K., *Fault Tolerant Computing Theory and Techniques*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[175] Ramakrishnan, T. and L. Kinney, "Extension of the Critical Path Tracing Algorithm", *27th Design Automation Conference Proceedings*, June 1990, pp. 720-723.

[176] Rimen, M. and J. Ohlsson, "A Study of the Error Behavior of a 32-bit RISC Subjected to Simulated Transient Fault Injection," *International Test Conference Proceedings*, September 1992, pp. 696-704.

[177] Rimen, M., J. Ohlsson, J. Karlsson, E. Jenn, and J. Arlat," Design Guidelines of a VHDL-based Simulation Tool for the Validation of Fault Tolerance," *Technical Report 162, Department of Comp. Engineering*, Chalmers University of Technology, 1993.

[178] Rogers, W. A., J. F. Guzolek, and J. A. Abraham, "Concurrent Hierarchical Fault Simulation: A Performance Model and Two Optimizations," *IEEE Transactions on CAD*, Vol. 6, No. 5, September 1987, pp. 848-862.

[179] Rogers, W. A. and J. A. Abraham, "CHIEFS: a concurrent hierarchical and extensible fault simulator," *International Test Conference Proceedings*, November 1985, pp. 710-716.

[180] Rogers, W. A. and J. A. Abraham, "High-Level Hierarchical Fault Simulation Techniques," *ACM Spring Computer Science Conference Proceedings*, 1985, pp. 89-97.

[181] Roth, J. P., "Diagnosis of automate failures: A calculus and a method," *IBM Journal of Research and Development*, Vol. 10, No. 7, July 1966, pp. 278-291.

[182] Roth, J. P., W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 10, October 1967, pp. 567-579.

[183] Rudnick, E. M., T. M. Niermann, and J. H. Patel, "Methods for Reducing Events in Sequential Circuit Fault Simulation," *International Conference on Computer-Aided Design Proceedings*, November 1991, pp. 546-549.

[184] Saab, I. D. and N. Hajj, "Parallel and Concurrent Fault Simulation of MOS Circuits," *International Conference on Computer Design Proceedings*, Oct. 1984, pp. 752-756.

[185] Schuler, D. M., T. E. Baker, R. S. Fisher, SS. Hirshhorn, M. B. Hommel, H. J. McGinness, and R.V. Bosslet, "A Program for the Simulation and Concurrent Fault Simulation of Digital Circuits Described with Gate and Functional Models," *1979 IEEE Test Conference Proceedings*, 1979, pp. 203-207.

[186] Schuler, D. M and R. K. Cleghorn, "An Efficient Method of Fault Simulation for Digital Circuit Modeled from Boolean Gates and Memories," *14th Design Automation Conference Proceedings*, June 1977, pp. 230-238.

[187] Schulz, M. H., and D. Pellkofer, "A Three-Valued Fast Fault Simulator for Scan-Based VLSI-Logic," *1st European Test Conference Proceedings*, April 1989, pp. 41-48

[188] Schulz, M. H., E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Pattern Generation System," *IEEE Transactions on Computer Aided Design*, Vol. 7, No. 1, January 1988, pp. 126-137.

[189] Schuster, M. D. and R. E. Bryant, "Concurrent Fault Simulation of MOS Digital Circuits," *Proceedings of Conference on Advanced Research in VLSI*, Massachusetts Institute of Technology, MA, January 1984.

[190] Seshu, S., "On an Improved Diagnosis Program," IEEE Transactions on Electronic Computers, Vol. EC-14, No. 1, February 1965, pp. 76-79.

[191] Seth, S. C. and V. D. Agrawal, "Forecasting Reject Rate of Tested LSI Chips," *IEEE Electron Device Letters*, Vol. EDL-2, No. 11, November 1981, pp. 286-287.

[192] Seth, S. C., V. D. Agrawal, and H. Farhat, "A theory of testability with applications to fault coverage analysis," *Proceedings of 1st European Test Conference*, Paris, France, April 1989, pp. 139-143.

[193] Seth, S. C., V. D. Agrawal, and H. Farhat, "A Statistical Theory of Digital Circuit Testability," *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990, pp. 582-586.

[194] Shadfar, M., A. Peymandoust, and S. Navabi, "Using VHDL Critical Path Tracing Models for Pseudo Random Test Generation", *VIUF Conference Proceedings*, April 1995, pp. 4.1-4.10.

[195] Shih, H., J. T. Rahmeh, and J. A. Abraham, "An MOS Fault Simulator with Timing Information," *International Conference on Computer Aided Design*, November 1985, pp. 45-47.

[196] Shih, H. C, J. T. Rahmeh, and J. A. Abraham, "FAUST: An MOS Fault Simulator with Timing Information," *IEEE Transaction on Computer-Aided Design*, Vol. CAD-5, No. 4, October, 1986, pp. 557-563.

[197] Silberman, G. M. and I. Spillinger, "Test Generation Using Functional Fault Simulation and the Difference Fault Model", *International Test Conference Proceedings*, September 1987, pp. 400-407.

[198] Silberman, G. M. and I. Spillinger, "The Difference Fault Model - Using Functional Fault Simulation to Obtain Implementation Fault Coverage", *International Test Conference Proceedings*, September 1986, pp. 332-339.

[199] Smith, J. W., K. S. Smith, and R. J. Smith, II, "Faster Architectural Simulation Through Parallelism ", *24th Design Automation Conference Proceedings*, June 1987, pp. 189-194.

[200] Smith, L. T. and R. R. Rezac, "A Simulation Engine in the Design Environment --- Part 2: Fault Simulation Methodology and Results," *VLSI Design*, Vol. 5, No. 12, December 1984, pp. 74-80.

[201] Smith, S. and R. von Blucher, "A Demand Driven Multi-Word Parallel Fault Simulator," *International Conference on Computer Aided Design Proceedings*, November 1987, pp. 6-9.

[202] Smith, S. P. and M. R. Mercer, "D³FS: A Demand Driven Deductive Fault Simulator", *International Test Conference Proceedings*, September 1988, pp. 582-592.

[203] Smith, S. P., M. R. Mercer, and B. Brock, "Demand Driven Simulation: BACKSIM ", *24th Design Automation Conference Proceedings*, June 1987, pp. 181-187.

[204] Son, K., "Fault Simulation with the Parallel Value List Algorithm," *VLSI Systems Design*, Vol VI, No. 12, December 1985.

[205] Song, O. and P. R. Menon, "Parallel Pattern Fault Simulation Based on Stem Faults in Combinational Circuits", *International Test Conference Proceedings*, September 1990, pp. 706-711.

[206] Stein, A. J., D. G. Saab, and I. N. Hajj, "A Special Purpose Architecture for Concurrent Fault Simulation," *International Conference on Computer Design*, October 1986, pp. 243-246.

[207] Synopsis, "Design for Test Data Sheet," Product Literature, 1995.

[208] Takahashi, N., N. Ishiura, and S. Yajima, "Fault Simulation for Multiple Faults by Boolean Function Manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 4, April 1994, pp.531-535.

[209] Thomas, J. J., "Automated Diagnostic Test Programs for Digital Networks," *Computer Design*, August 1981, pp. 63-67.

[210] Torku, K. and C. Radke, "Quality Level and Fault Coverage for Multichip Modules," *20th Design Automation Conference*, 1983, pp. 201-206.

[211] Thompson, E. W. and S. A. Szygenda, "Digital Logic Simulation in a Time-based, Table-Driven Environment, Part, Parallel Fault Simulation," *IEEE Computer*, Vol. 8, No. 3, pp. 38-40, 1975.

[212] TransEDA Limited, "VHDL Cover," Product Literature, 1995.

[213] Tsiang, S. H., and W. Ulrich, "Automatic Trouble Diagnosis of Complex Logic Circuits," *Bell System Technical Journal*, Vol 61, No. 4, pp. 1177-1200.

[214] Ulrich, E., "Concurrent Simulation at the Switch, Gate and Register Levels," *International Test Conference Proceedings*, November 1985, pp. 703-709.

[215] Ulrich E. G., "The Evaluation of Digital Diagnostic Programs Through Digital Simulation," *Computer Technology, IEEE Conference Publication No. 32*, 1967, pp. 9-19.

[216] Ulrich, E. G. and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks," *10th Design Automation Conference Proceedings*, 1973, pp. 145-150.

[217] Ulrich, E, D. Lacy, N. Phillips, J. Tellier, M. Kearney, T. Elkind, and R. Beaven, "High Speed Concurrent Fault Simulation with Vectors and Scalars," *17th Design Automation Conference Proceedings*, June 1980, pp. 374-380.

[218] Underwood, B. and J. Ferguson, "The Parallel-Test-Detect Fault Simulation Algorithm ", *International Test Conference Proceedings*, August 1989, pp. 712-717.

[219] van der Linden, J. Th., M. H. Konijnenburg, "Parallel Pattern Fast Fault Simulation for Three-State Circuits and Bidirectional I/O", *International Test Conference Proceedings*, October 1994, pp. 604-613.

[220] Vandris, E. and G. Sobelman, "Algorithms for Fast, Memory Efficient Switch-Level Fault Simulation", *28th Design Automation Conference Proceedings*, June 1991, pp. 138-143.

[221] Vandris, E., and G. Sobelman, "Fast Switch-Level Fault Simulation Using Functional Fault Modeling," *International Conference on Computer Aided Design Proceedings*, 1990, pp. 74-77.

[222] Wadsack, R. L., "Fault Coverage in Digital Integrated Circuits," *Bell System Technical Journal*, Vol 57, May-June 1978, pp. 1475-1488.

[223] Wadsack, R. L., "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *Bell System Technical Journal*, Vol 57, May-June 1978, pp. 1449-1474.

[224] Ward, P. C. and J. R. Armstrong, "Behavioral Fault Simulation in VHDL ", *27th Design Automation Conference Proceedings*, June 1990, pp. 587-593.

[225] Waicukauski, J. A., E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy, "A Statistical Calculation of Fault Detection Probabilities by Fast Fault Simulation," *International Test Conference Proceedings*, November 1985, pp 779-784.

[226] Waicukauski, J. A., E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy, "Fault Simulation for Structured VLSI", *VLSI Systems Design*, December 1985, pp. 20-32.

[227] Waicukauski, J. A., V. P. Gupta, and S. T. Patel, "Diagnosis of BIST Failures by PPFSP Simulation ", *International Test Conference Proceedings*, September 1987, pp. 480-484.

[228] Waicukauski, J. A., E. Lindbloom, E. B. Eichelberger, and O. P. Forlenza, "A Method for Generating Weighted Random Test Patterns," *IBM Journal of Research and Development*, Vol. 33, No. 2, March 1989, pp. 149-159.

[229] Waicukauski, J. A., E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition Fault Simulation," *IEEE Design and Test of Computers*, Vol. 4, No. 2, April 1987, pp. 32-38.

[230] Wang, D. T., "An Algorithm for the Generation of Test Sets for Combination Logic Networks," *IEEE Transactions on Computers*, Vol. C-24, No. 7, July 1975, pp. 742-746.

[231] Wang, X., F. J. Hill, and Z. Mi, "A Sequential Circuit Fault Simulation by Surrogate Fault Propagation", *International Test Conference Proceedings*, August 1989, pp. 9-18.

[232] Wang, W., K. S. Trivedi, B. V. Shah, and J. A. Profeta III, "The Impact of Fault Expansion on the Internal Estimate for Fault Detection Coverage", *24th International Symposium on Fault-Tolerant Computing*, June 1994, pp. 330-337.

{233] Williams, T. W. and N. C. Brown, "Defect Level as a Function of Fault Coverage," *IEEEE Transactions on Computers*, Vol. C-30, No. 12, December 1981, pp. 987-988.

[234] Wilmot, A. R., "Integrating VHDL and WAVES to Form a Design Test Environment", *VHDL Users' Group Conference*, April 1991, pp. 23-27.

[235] Yetter, I. H, "High-speed fault simulation for the Univac 1107 computer system," *Proceedings of the ACM national conference*, 1968, pp. 265-277.

[236] ZyCAD, "Paradign XP Product News," Product Literature, 1995.

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

   a. Conducts vigorous research, development and test programs in all applicable technologies;

   b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

   c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

   d. Promotes transfer of technology to the private sector;

   e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.